HPI Hasso Plattner Institut
IT Systems Engineering
Universität Potsdam

Master's Thesis

Language Identification Using Deep Convolutional Recurrent Neural Networks

<Deutscher Titel bei englischer Arbeit>

Tom Herold

tom.herold@student.hpi.uni-potsdam.de

XX.0X.2017

Supervisors
Prof. Dr. Christoph Meinel
Dr. Haojin Yang

Internet Technologies and Systems
Hasso Plattner Institute
University of Potsdam, Germany

Abstract

Zusammenfassung

Acknowledgments

I would like to express my gratitude to my supervisors Dr. Haojin Yang and Prof. Dr. Christoph Meinel. I want to thank them for giving me the opportunity to research this interesting topic, as well as for their guidance and advice. I especially want to thank Dr. Yang for his insights, support, and inspiration regarding the deep learning techniques used in my masters thesis. I would also like to thank my colleagues Georg Wiese, Christian Bartz, Tom Bocklich, Norman Rzepka, Christoph Sterz, and Johannes Jasper for the many fruitful discussions about language identification and machine learning in general. I owe them a great deal of gratitude for supporting and encouraging me while working on this thesis. I am also very grateful for the stylistic critique, review and language support by Christoph Sterz and Patrick LÃijhne.

Thank you.

Contents

Abbreviations

ASR automatic speech recognition

CNN convolutional neural network

CRNN convolutional recurrent neural network

FC fully connected layer

GPU graphics processing unit

LID language identification

LSTM long short-term memory

NIST National Institute of Standards and Technology

ReLU Rectified Linear Unit

RNN recurrent neural network

SVM support vector machine

t-SNE t-distributed stochastic neighbor embedding

1 Introduction

Computers have become ubiquitous in our daily lives. Millions of people around the world use speech input to conveniently interact with their devices. Speech input excels in certain situations, such as hands-free interaction, when driving or as a efficient alternative to text input instead of typing on small screens. In all these interactions, the first step to understand users is to correctly identify their input language. In some settings, computers might well use metadata such as known locations or system default languages for this task. Yet, often times it is much preferred to infer a language directly from the speech input. This can be the case in countries with more than one official language or for multilingual users.

In the same timeframe of this development, deep learning and artificial neural networks have become the state of the art for many pattern recognition problems. Deep convolutional networks have become the best-performing method for solving various computer vision tasks, such as image classification[1], object detection[1, 2], text detection[3, 4], semantic segmentation[5, 6], object tracking[7], image retrieval[8], and many others. The task of automatic language identification, however, was previously mainly done through classical machine learning algorithm approaches which relied on domain-specific expert knowledge in the field of audio signal processing.

In this thesis we propose a novel system for language identification using deep learning techniques. We use convolutional recurrent neural networks to directly classify languages on a given speech input. We benefit from the aforementioned advances in the computer vision community, by transferring the language identification problem from the audio context to the image domain.

1.1 Contributions

In this thesis, we present a novel approach to language identification systems using deep learning techniques. For this, we transfer the given audio classification problem into an image-based task to apply image recognition algorithms on the transformed data. Our contributions can be summarized as follows:

- We investigate the suitability of *convolutional neural networks* (CNN) for the task of language identification. As a solution to this challenge, we propose a hybrid network, combining the descriptive powers of convolutional neural networks with the ability of *recurrent neural networks* (RNN) to capture temporal features. This approach is called *convolutional recurrent neural network* (CRNN).

- We implement a CNN and a CRNN system in Python using the deep learning frameworks Keras and TensorFlow. We show that the CRNN approach outperforms all other methods in every single evaluation with respect to accuracy and F1 score.

- To train our system, we compile our own large-scale dataset of audio recordings. We explain how we obtain and process more than a thousand hours of suitable human speech recording for our task.

- We assess several machine learning metrics on our system with respect to our test data. Furthermore, we investigate the influence of noisy environments on our system. We discuss the system's ability to differentiate between several languages and extend the system to even more languages.

- To showcase our system, we develop a web service demo application employing our best-performing model, which is published for use by others.

1.2 Outline of the Thesis

This thesis is structured as follows. In Chapter 2, we introduce the language identification problem and state our research hypotheses. Chapter 3 explains the theoretical background of the deep learning techniques and algorithms used in this thesis. Chapter 4 introduces related work and alternative approaches to the language identification task (LID). In Chapter 5, we describe the audio datasets we collected for training and evaluating our system. Implementation details are outlined in Chapter 6. Further, we describe the network architectures of our models. Evaluation results are reported and discussed in Chapter 7 and followed up by various experiments for assessing the robustness of our system to music and noise. In Chapter 8, we propose a web service to showcase a potential use case for language identification. Finally, we close this thesis by summarizing all our observations in Chapter 9 and outline future work.

2 The Language Identification Problem

Automatic language identification (LID) is the process of determining the language spoken in an audio recording. The language identification systems proposed in this thesis consume audio recordings and use deep learning techniques to perform an automated classification of the recordings' source language. Language identification is often the first step in a spoken-language processing pipelines. Automatic language identification systems are employed by a wide variety of applications from industry and research to entertainment. In the following, multiple examples for language identification systems are presented.

2.1 Language Identification as the Key to Speech Tasks

In contrast to LID systems, *automatic speech recognition* (ASR) is the task of transcribing spoken language into readable text, sometimes also known as *speech-to-text systems*. Selecting the correct input language for these systems is crucial for transcribing single letters into meaningful words and adhering to the correct grammar. Many commercially available ASR system require manually setting the input language and would greatly benefit from an automated language identification system as part of their processing pipeline.

Many call center operators benefit from LID systems by automatically routing telephone calls in order to connect a caller with a suitable native speaker. This approach is used both for customer care hotlines (connecting callers with help desk agents) as well as government agencies, such as emergency telephone services. Muthusamy et al. report that manually matching emergency calls to US law enforcement agencies with native speakers involves a significant delay of up to three minutes. Automatic language identification systems speed up this process and efficiently support human agents [9].

Similar to language identification, some research is focused on dialect detection. To foster research in this field, DARPA established the TIMIT dataset for dialect identification of North American speakers [10]. The dataset features eight major North American dialects and has long been the default corpus for comparative research on LID systems. Recently, Germany's Federal Office for Migration and Refugees (BAMF) announced plans for using dialect identification systems as an additional resource in identifying a person's origin.[1]

Language identification is also the first step in automated translation tasks. At the time of writing this thesis, we discovered that not even the Google Translate mobile app has automated language detection to determine the input language for speech input. Automated input language detection is available for written texts, but Google Translate's

[1] `http://www.theverge.com/2017/3/17/14956532/germany-refugee-voice-analysis-dialect-speech-software`, accessed 13 April 2017

voice input feature is only available for use after manually selecting an input language. We believe that having an automated language detection system is extremely helpful to users.

An ever-increasing number of people connect to the internet or communicate with each other using their smartphones. While touchscreen input is complicated in some situations, the demand for voice interaction grows steadily. Many tasks can already be solved through voice input. This hands-free, voice-enabled interaction is a trend that we see in other industries such as automative computing as well.

Automatic LID systems and machine intelligence are also helpful for some entertainment companies. When we started working on this thesis, we were briefly inspired by the challenges of the Berlin-based start-up Dubsmash.[2] Their app lets users create a mash-up of a large variety of existing songs, movie quotes, or other voice snippets with a ten-second video recording of the user. The resulting video clip can be shared with friends and usually features a funny and personal reinterpretation of the audio source's original context. The success of the app is directly proportional to users' interest in the offered sound clips. In some cases, however, users are offered clips in foreign languages, which are often perceived as not funny or inappropriate. In this case, a LID system could be used to classify the language of the millions of audio snippets available in their library. Consequently, only sounds in the users' native languages could be recommended to the users.

2.2 Task Specification in This Thesis

In this thesis, we propose a language identification system for classifying the languages of given audio recordings. Our system is trained using human voice recordings. We evaluate the suitability of convolutional neural networks for a LID system. Finally, this approach is extended with a recurrent neural network to compose a hybrid network known as *convolutional recurrent neural network* (CRNN). The system's performance is evaluated on a set of news broadcasts and speeches made by members of the European Parliament. We further assess the system's robustness to noisy environments and background music. This thesis states the following hypotheses:

1. Convolutional neural networks are an effective, high-accuracy solution for language identification tasks.

2. Spectrogram images are a suitable input representation for learning audio features.

3. Convolutional recurrent neural networks improve the classification accuracy for our LID task compared to a plain, CNN-based approach.

[2] `https://www.dubsmash.com/`, accessed 13 April 2017

3 Theoretical Background

In this chapter, we introduce the theoretical background of the machine learning algorithms used throughout this thesis. We explain the different machine learning concepts and the purpose of *classifiers*. Furthermore, we lay the foundations for the individual building blocks and layers of deep neural network systems. We explain the differences between *convolutional neural networks* (CNN) and *recurrent neural networks* (RNN) and continue by describing hybrid models composed of both of these. More specifically, we describe the *convolutional recurrent neural network* (CRNN) architecture as used in this thesis. Finally, we present different representations of audio data suitable for machine learning tasks.

3.1 Machine Learning

Machine learning is a subfield of computer science that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning relies on mathematical algorithms and statistics to find and learn patterns in data.

3.1.1 Types of Machine Learning

The field of machine learning covers a multitude of different learning algorithms. While each of these is based on a different mathematical foundation, they also serve many different purposes. *Classification* algorithms divide input data into two or more distinct classes. Each new input sample can then be assigned to one of these learned classes. For example, we could imagine an app that automatically classifies pictures of food and recognizes and names the respective dishes. While classification results are always discrete values, a *regression* algorithm outputs continuous values instead. An example is a regressor predicting the future price of your favorite food based on historical price data. For other purposes, it is more important to know which data samples are similar to each other and form a group of their own. *Clustering* algorithms divide data into groups and, unlike with classification, these group are usually not known before. For instance, a customer management system could cluster everyone into distinct clusters of different focus groups.

Regardless of an algorithm's purpose, machine learning typically can be divided into three categories from a high-level point of view:

Supervised Learning is the task of building a machine learning model with labeled training data. Every data sample used during training is a pair consisting of a vector or matrix representation of the data and a label identifying the data as belonging to a certain class. Typically, a label is represented as a single number or a one-hot-encoded vector. Supervised learning algorithms learn an inference function mapping every data sample to its expected output label. A trained model should then be able to infer a suitable output class or value for new unknown data.

Unsupervised Learning is the task of building a machine learning model with unlabeled training data. Unlike with supervised learning, none of the training samples include labels of the desired model outcome. Unsupervised learning algorithms learn a function by detecting the hidden structures inside the data. Many clustering algorithms fall into this category.

Reinforcement Learning is the task of building a machine learning model without a set of classical training data. Instead, a reinforcement learning system is set within a specific environment and executes a set of actions. Each action's effect on the environment is measured, and reward is calculated. By maximizing this reward, the system finds actions that are most effective towards achieving the specified task. This setup is similar to computer simulations in other problem domains.

The system described in this thesis is a supervised classification approach using a large-scale labeled training dataset.

3.1.2 Classification

Classification in the context of machine learning refers to the task of assigning a data sample to the corresponding class. Classification algorithms are a form of supervised learning and, hence, need a training dataset of labeled data. The input to classifiers is referred to as *features*. Classical machine learning algorithms, such as *support vector machines* (SVM) and linear classifiers, usually require input features carefully designed by domain experts as a good representation of the original problem. This process is often referred to as *feature engineering*. In contrast, deep learning classifiers such as *neural networks* are able to directly work on the raw data representations. This has the benefit of building machine learning systems without the need for handcrafted features of domain experts but comes at the price of increased computational requirements. For computer vision tasks, for example, it used to be customary to train classical systems on preprocessed and extracted image features, such as SIFT key points [11] or HOG descriptors [12]. Deep learning systems, on the other hand, are capable of processing all the original raw pixels of the input image.

Figure 1 shows an example of two binary classifiers dividing a simple dataset into two classes in 2D space. On the left-hand side is linear classifier dividing all points along a

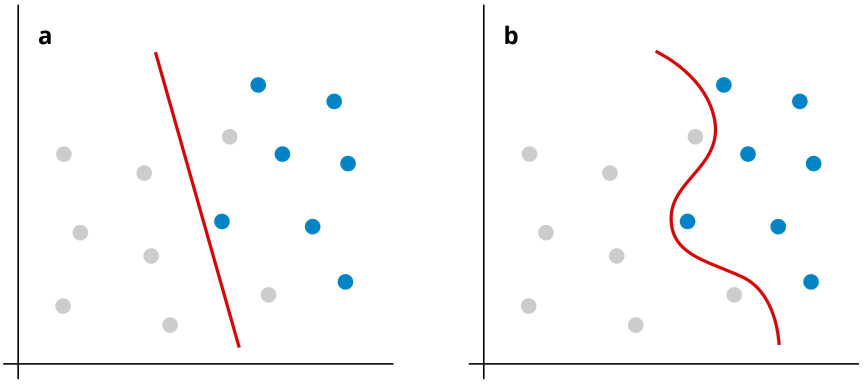


Figure 1: An example of two classifiers: A *linear classifier* (a) divides the data using a straight line but misclassifies two data points. A *neural network* (b) is able to learn a more complex decision boundary and separates the dataset without errors.

straight line. While this sort of classifier is very easy to train and understand it lacks the necessary foundation to divide more complex datasets. On the right-hand side is a neural network (more details in the next section) featuring a more complex, yet more accurate decision boundary.

3.2 Building Blocks of Deep Neural Networks

Modern deep learning systems are composed in a layer-wise fashion. Each layer performs a nonlinear computation to extract features and learns a representation of the input data before passing on its outputs to the next layer in the architecture. The term *deep learning* itself is not clearly defined but usually refers to having an at least two-layered model. Typical state-of-the-art systems have more than ten layers, some recent publications are even going as deep as 152 layers [13]. While the total number of layers is, by no means, an indicator for an accurate and precise model, it makes capturing a larger and more generalized data representation possible.

3.2.1 Neural Networks and Fully-Connected Layers

While early *neural networks* were inspired and named after discoveries in biology, particularly the neurons in animal brains, modern interpretations treat *artificial neural networks* as a mathematical model of interconnected artificial neurons or a series of matrix operations [14, 15].

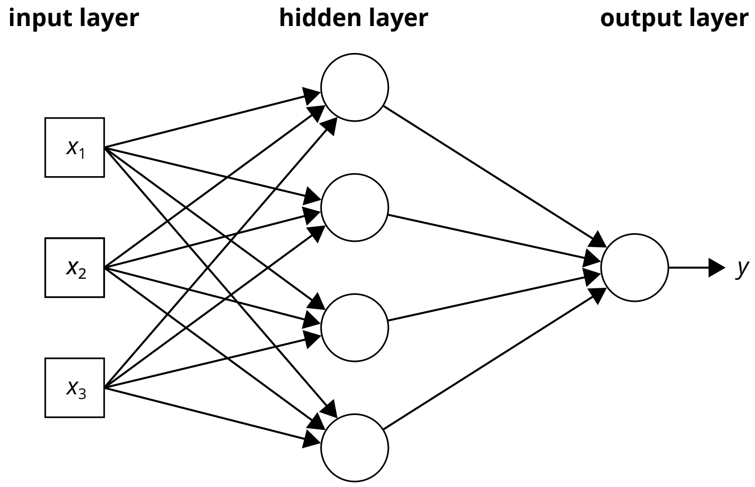


Figure 2: A neural network with three layers. Each layer is connected to all outputs of the previous layer. All central layers, which are neither connected to the network's inputs or output, are referred to as *hidden layers*. Note that all connections between the neurons are directed and do not include self-references. Therefore, we refer to these classes of models as *feedforward neural networks*.

Feedforward neural networks are designed as a chain of layers, each applying an affine transformation. Each layer uses the output of the previous layer as an input, and all connections are directed without any self-references, $f(\boldsymbol{x}) = f_i(f_{i-1}(...(f_1(\boldsymbol{x}))))$, where $\boldsymbol{x} \in \mathbb{R}^n$ is an input vector. The first and last layer are referred to as the *input* and *output* layer, respectively.

When fed with an input vector x, a neural network computes an affine transformation:

$$f(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \sigma(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b})$$

where σ is function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$, $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ an $n \times m$ *weight matrix*, and $\boldsymbol{b} \in \mathbb{R}^m$ a *bias term* [16, p. 192]. Both the weights and biases are trainable parameters and updated during the learning process. By itself, the chain of linear transformations can only model linear relations. To model more complex scenarios, σ is a nonlinear *activation function*. Typical candidates for activation functions in neural networks are *rectified linear units* (ReLU) [17]:

$$\sigma(\boldsymbol{x}) = \max(\boldsymbol{x}, 0)$$

Within the deep learning community, neural networks are also referred to as *fully-connected* layers (FC), owing to the fact that each artificial neuron is connected to all outputs of the previous layer. Figure 2 shows an example of a three-layer neural network. Any layer not directly connected to the input or output is referred to as a *hidden layer*. In our example, we only have a single output neuron and, hence, a binary classifier for the two classes.

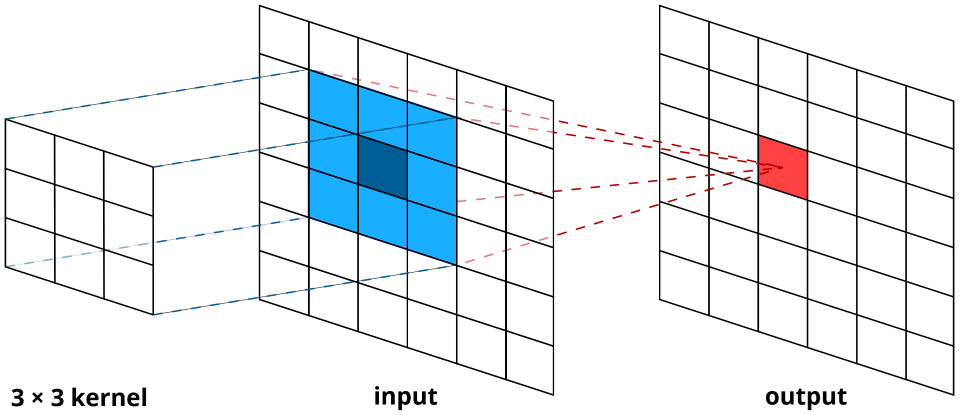


Figure 3: A convolution operation on an image using a 3 × 3 kernel. Each pixel in the output image is the weighted sum of 9 pixels in the input image. The weights are not fixed but instead learned by the model.

For multi-class tasks, we can add more output neurons. Typically, fully-connected layers are used as final layers of a deep neural network architecture serving as a classifier.

3.2.2 Convolutional Layers

Convolutional neural networks (CNN) are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. CNN specialize in processing data that has grid-like structure, such as the 2D grid of pixels of an input image as in our case. As the name suggests, they make use of the mathematical *convolution* operation. For computer vision tasks this can be thought of as the weighted sum of all pixels in a neighborhood. More formally, convolution layers are expressed as [18]:

$$\boldsymbol{Y}^{j(r)} = \sigma\left(\boldsymbol{b}^{j(r)} + \sum_i \boldsymbol{K}^{ij(r)} * \boldsymbol{X}^{i(r)}\right)$$

where $\boldsymbol{X}^i$ and $\boldsymbol{Y}^j$ are the i-th input and j-th output map, respectively. $\boldsymbol{K}^{ij}$ defines the kernel between those two maps and $*$ denotes the convolutional operation. σ is an activation function, such as ReLU. $\boldsymbol{b}^j$ is the bias of the j-th output map and r indicates a local region where weights are shared.

Typical kernels for CNNs have a square size of 3×3. Thus, for each pixel in the output map a neighborhood of $3 \times 3 = 9$ pixels is evaluated by the convolutional operation. Figure 3 shows the convolution of a single pixel in the output map $\boldsymbol{Y}^j$. All pixels involved in the calculation of this pixel are referred to as its *receptive field*.

Kernels are learned during the training and can be thought of feature detectors. This concept is often used in the field of image processing where predefined kernels such as a Sobel filters are used for edge detection and Gaussian filters for blurring. Similarly, a CNN learns to detect simple features such as edges in lower layers. Convolutional layers at the end of the network architecture combine these simple features to compound shaped features able to reliably identify an input.

Typically, a CNN does not only learn one kernel but several, one for each desired output dimension. Other hyperparameters include the *stride* and amount of *zero padding*. The former defines by how many pixels the kernel is translated on the input. The latter specifies how to handle convolutions along the edge of an input where the kernel would need to include pixels from outside the image. Similarly to fully-connected layers, an nonlinear activation function is applied on the output of a convolutional layer as well.

Unlike fully-connected artificial neural networks, CNNs need fewer parameters through the use of *weight sharing*. A fully-connected neural network of m inputs and n outputs requires $m \times n$ weight parameters. A convolutional layer, on the other hand, each member of the kernel is used on every position of the input. Instead of learning parameters for every position in the input is only uses one set of $k \times k \times c \times o$ weights, where k is the kernel size, c the number of channels in the input, and o the number of desired output channels (often called feature maps). For instance, a $100 \times 100 \times 3$ RGB color input image would require $100 \times 100 \times 3 \times 128 = 3840000$ weight parameters for a fully-connected layer with 128 outputs. In contrast, a CNN only uses $3 \times 3 \times 3 \times 128 = 3456$ weight parameters for a two-dimensional kernel of size 3×3 and for the same number of output units.

3.2.3 Pooling Layers

A convolutional layer is typically followed by a *pooling* layer (sometimes also called subsampling or downsampling layer). Pooling layers are used reduced the spatial dimensions (width and height) on the input data. *Max pooling*, for instance, only outputs the maximum value within its neighborhood. Similar to convolutions, pooling uses a $n\times$ kernel parameter to determine which input pixels are effected by the subsampling.

While pooling leads to a reduction in information, this has two main benefits. First, pooling improves the computing efficiency of a network since next layers in the network architecture need to process less data. Typically, 2×2 pooling with a stride of 2 is used, effectively halving every input along the width and height dimension. Second, pooling helps against overfitting.

Figure 4 shows an example of a typical convolutional neural network constructed from convolutional, pooling and fully-connected layers. Typically, an alternating sequence of convolutional layers followed by pooling layers is finalized by a fully-connected layer serving as a classifier.

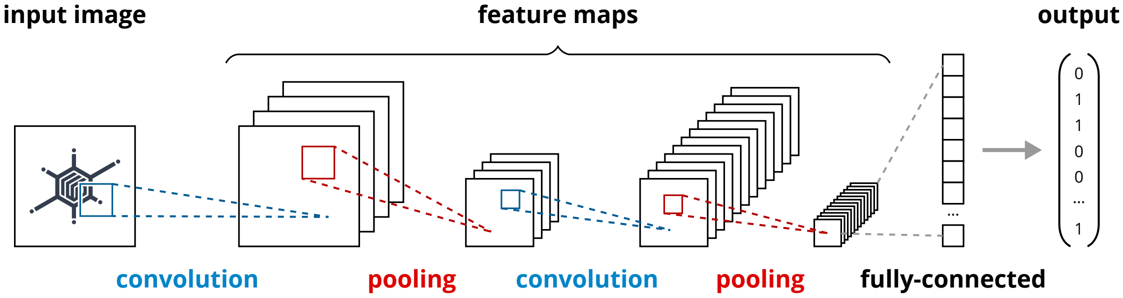


Figure 4: A typical *convolutional neural network* architecture. An input image is fed through a series of convolutional and pooling layers. Each convolution extracts higher-level features and increases the number of feature maps. Each pooling layer subsamples the data, typically reducing the x and y dimension by half. A final, fully-connected layer serves as a classifier to predict the output value.

3.2.4 Softmax Cross-Entropy Loss Function

The *softmax* function is an activation function applied to last layer to obtain probability distribution. It maps any $K \in \mathbb{N}^{+}$-dimensional input vector $\boldsymbol{x} \in \mathbb{R}^{K}$ onto a vector with a $[0, 1]$ value distribution, where the sum of all entries is 1:

$$\text{softmax}(\, x\,)_i = \frac{\exp^{x_i}}{\sum_{k=0}^{K} \exp^{x_k}}$$

In order to train a neural network we constantly calculate the difference between a desired output class y and a predicted class $\hat{y}$. We refer to such an error measure between the approximated and expected output value as *loss function*. A commonly applied loss function for multi-class classification problem, as presented in this thesis, is cross-entropy loss:

$$L_{cross\_entropy} = -log\left(\frac{\exp^{f_y}}{\sum_k \exp^{f_k}}\right)$$

where f_y is the probability granted to the correct class and f_k are probabilities of all classes. Since part of the formula contains a softmax calculation we refer to this loss function as *softmax cross-entropy loss*.

3.2.5 Back Propagation Through Time

3.3 Recurrent Neural Networks

3.3.1 Long Short Term Memory Networks

3.4 Convolutional Recurrent Neural Networks

While both CNNs and RNNs can be used completely on their own, for some task it is worthwhile to use a combination of both. These *hybrid networks* can combine the best of both worlds. Specifically, in this thesis we are using *convolutional recurrent neural network* consisting of a CNN part and an RNN part. Tang et al. first introduced this architecture in the context of document classification [19].

The convolutional part is responsible for detecting and extracting local visual features into an intermediary audio representation. A recurrent part is tasked with temporal summarization of these features taking the global structure into account. Figure 5 show the architecture of a CRNN network. The last convolutional layer of a CNN is replace with a LSTM layer. In the context of language identification, the outputs of the last convolutional layer result in a tensor $t \times f \times c$, where t are time steps, f the frequency features, and c the feature map outputs of the CNN. This tensor is stacked along the frequency dimension and split along the time axis into t time step inputs of $f \times c$ for the LSTM.

For this thesis we use a bidirectional LSTM that performs both forward and backward pass a long the time sequence. We combine that last output of the forward pass with the first output of backward into a single, concatenated output. Finally, this is fed into a fully-connected layer serving as a classifier.

3.5 Audio Representations

There are many different representations for audio data. Many people are familiar with waveform representation which is usually represented as a plot of signal amplitude along a time axis. A raw waveform representation is typically not directly used as input to machine learning systems. Other representations such as *spectrogram* images or *mel-frequency cepstral coefficients* (MFCCs) vectors are more common. To obtain both of these we need to transform the original audio signal.

First, we discretize the audio signal into into a list *spectral vectors* (a spectrum) by applying a Fourier transformation. Each spectral vector consists of amplitude and frequencies pairs

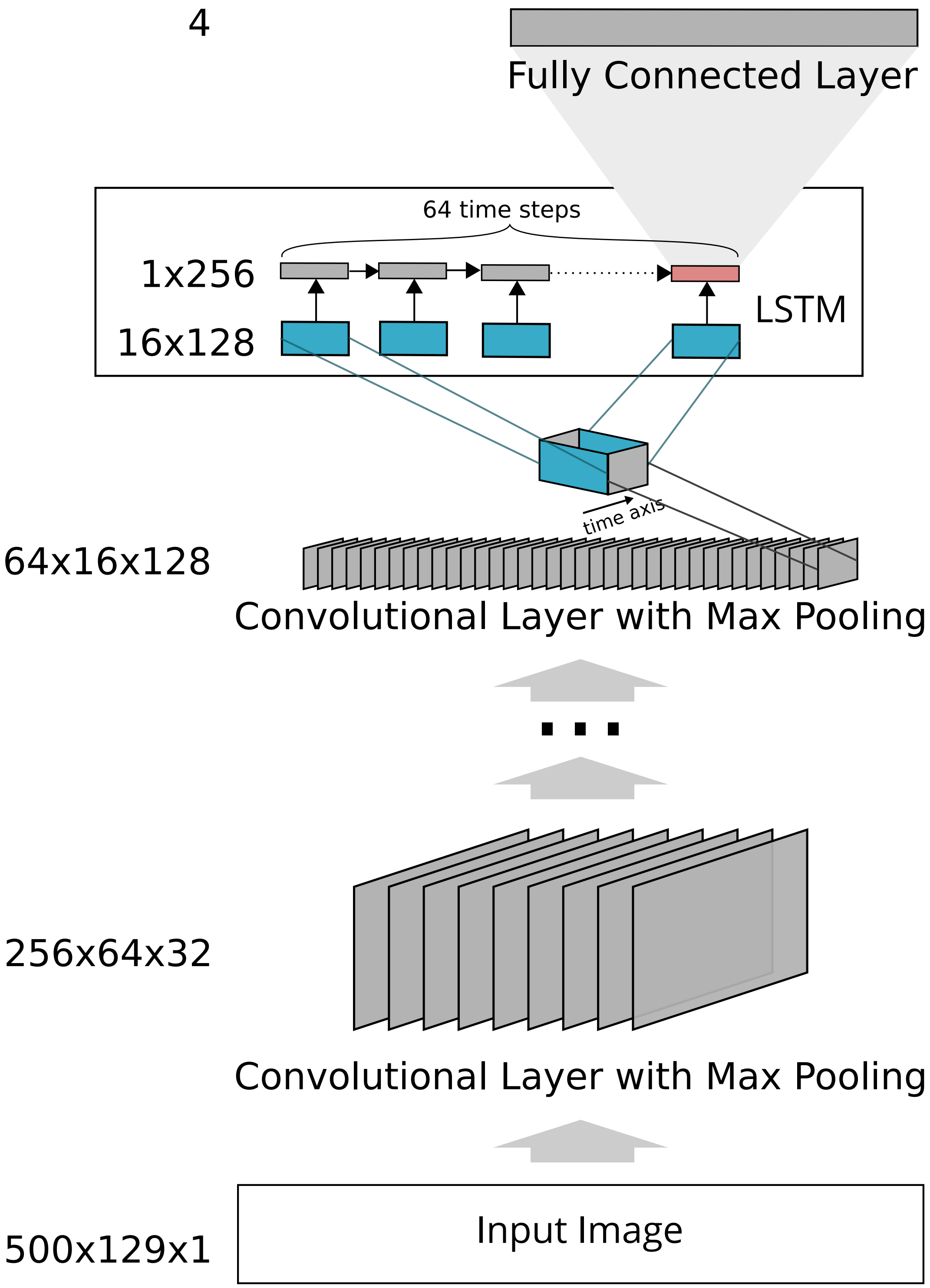


Figure 5: Our proposed CRNN hybrid network architecture consists of two parts. A CNN extracts local visual feature from our input images and outputs a intermediary representation of our audio frequencies. The output of the final convolutional layer is sliced along the time axis into time steps of frequency features for the LSTM. The output of the final LSTM time step is fed into a fully-connected layer for classification.

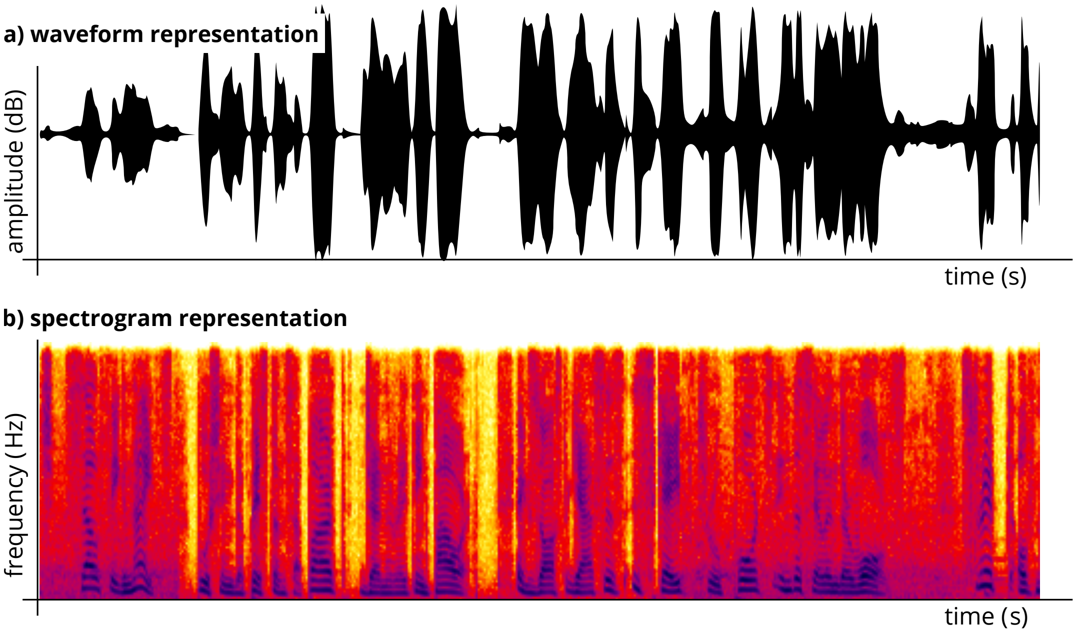


Figure 6: Different visual representations for a ten second audio file. a) shows a familiar waveform representation. b) shows the spectrum of frequencies known as spectrogram. Spectrogram images are a commonly used representation for audio learning tasks.

at a point in time. When visualizing these we get a spectrogram image as shown in Figure 6.

The time domain is along x axis and the frequency domain in Hz is along the y axis. Additionally, the color intensity for each data point represents the amplitude or loudness value for a given frequency at that point of time. Spectrograms are especially useful for studying time–frequency features such as *phonemes*. A phoneme is the smallest unit of sound expressing part of an word or single letters. While phonemes themselves are universally shared across different languages, their arrangement as a sequence forms the individual sounds and words distinctive to one language. Frequencies form phonemes which in turn form words. These can be combined to sentences or assemble the vocabulary of a language. Within a spectrogram image we can clearly see distinct, bright, ripple-like patters, each representing a heightened frequency activation at that time. Due to the anatomy of the human vocal cords we emit sounds vibrating on multiple frequencies while speaking. The frequency axis is divide into multiple *formants*, defined as a range of frequencies of a complex sound in which there is an absolute or relative maximum in the sound spectrum. There are several formants ranging form the lower speech frequencies to the higher ones. Voxels can be identified by their difference in the lower F1 and higher F2 format. Already, we note that there is a lot more information encoded into theses images then first meets the eye and, hence, a lot of speech processing systems use spectrograms.

MFCC is obtained by using linear cosine transform of log power spectrum on a nonlinear mel-frequency scale. This means that, first the powers of the spectrum are mapped onto the *mel-scale*. The mel-scale groups frequencies onto a perceptual scale of equally distanced audio pitch values [20]:

$$\boldsymbol{m} = 2595 \log_{10} \left(1 + \frac{f}{700}\right)$$

where $\boldsymbol{m}$ are the resulting mels and f frequencies in Hz. The mel-scale is modeled after human ear's deficiency to differentiate adjacent frequencies. Effectively this groups frequencies together into bins with many, smaller bins for lower frequencies and only a few larger for the higher frequencies. Further, we calculate the powers of the mel-spectrum and apply the logarithm to these. The resulting energies are fed into a *discrete cosine transformation* of which we keep the first 13 coefficients as our MFFC vector [21].

4 Related Work

In this chapter, we lay out related work concerning neural network designs in general and hybrid networks that are specifically tailored to language identification. Additionally, we highlight related work on input feature representations suitable for machine learning on audio files. Furthermore, we present research on *i-vector systems*, the traditional approach to LID. Finally, we list related work on data augmentation.

4.1 Convolutional Neural Network Architectures

With good results in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [22], AlexNet [23], VGGNet [24], and GoogLeNet/Inception [25] have become the de facto standards for neural network designs in the computer vision community.

The Visual Geometry Group at Oxford University published their deep neural network design, which they called VGGNet. Simonyan et al. proposed a convolutional neural network architecture with a depth of up to 16 or 19 weight layers [24, 26]. They presented a thorough evaluation of the recently increasing depth of neural networks by using convolutional layers with 3×3 kernel sizes and ReLU activations. Their system ended up winning the ILSVRC 2104 challenge.

Szegedy et al. reported on several iterations of Google's convolutional neural network architecture, called GoogLeNet or Inception [25, 27, 28]. These deep and very deep convolutional neural networks repeatedly set the state-of-the-art record for minimal classification errors in the ILSVRC competition. The introduced network architectures have multiple advantages over previous CNN designs such as VGGNet. Google introduced so-called *inception modules* or *mini-networks*, which aim to factorize convolutions with larger filter sizes in order to reduce training times and the number of model parameters. The idea is to replace larger spatial filters (for instance, 5×5 and 7×7), which are disproportionally expensive in terms of computation, with less expensive, smaller inception modules without any loss of visual expressiveness. The inception modules are represented as a sequence of 3×3 convolutions followed by a 1×1 convolution. In case of replacing 5×5 filters, the authors were able to achieve a computational speed-up of 28 %. The resulting Inception-v2 and Inception-v3 networks consist of 42 layers and are trained using the RMSProp optimizer [29]. Compared to the VGGNet-like networks, the inception networks feature lower overall computational costs, while offering higher accuracy on image vision tasks.

A second innovation introduced by inception networks is the use of a technique called *batch normalization* [30]. During training, the parameters of each layer continuously change and, hence, also the value distribution of the respectively next layer's input. This makes training deep neural networks particularly complicated and slows down the training phase by requiring lower learning rates and careful parameter initialization. Szegedy et al. refer to this phenomenon as *internal covariate shift*. Their proposed solution is to normalize the inputs of each mini-batch for the following layer to unit vectors. This results in a number of benefits. Foremost, the authors were able to drastically reduce the time needed for the models to converge. Second, batch normalization enabled them to use higher learning rates without running into the *vanishing* or *exploding gradient problem*. Furthermore, batch normalization acts as model regularizer positively affecting the generalization abilities of the network. In turn, this eliminates the need for dropout layers as regularizers. The authors conclude that by simply adding batch normalization to the convolutional layers of the inception network, they were able to beat the state of the art in the ILSVRC challenge.

Shi et al. proposed an end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition [31]. The authors developed a hybrid neural network consisting of a convolutional part and a recurrent part for *optical character recognition* (OCR). Convolutional layers are used as robust feature extractors for the input images. The resulting feature maps are interpreted as a time sequence of feature vectors, which is fed into a LSTM network to capture the contextual information within the sequence. The authors jointly trained the whole CRNN with a *connectionist temporal classification* loss function (CTC) [32] to output a sequence of letters. The best-performing network architecture is a *VGGNet*-based one composed of seven convolutional layers with max pooling followed by a bidirectional LSTM. The paper concludes that the use of batch normalization greatly reduces the training time. Additionally, the authors employed 1 × 2-sized rectangular pooling windows instead of conventional square ones. This tweak yielded feature maps of larger width and, hence, longer sequences for the RNN.

4.2 Spoken-Language Processing Systems

Much of the research on audio representations and spoken-language processing is rooted in various related research communities: *music information retrieval* (MIR), *automatic speech recognition* (ASR) and *language identification* (LID). Alongside the rising popularity of neural networks within the computer vision community, we can witness their use in audio-based tasks as well. Early LID system integrated and combined shallow neural networks within their i-vector systems (more details in Section 4.4) [33, 34, 35, 36]. These models usually feature no more than three layers and consist solely of classic neural networks or fully-connected layers. While these systems benefit neither from the more efficient computation of CNNs nor the expressiveness of deeper networks, they were already able to improve on existing systems.

Song et al. reported the use of an end-to-end-trainable, hybrid, convolutional recurrent neural network for automatic speech recognition [37]. The authors focused on classifying phoneme sequences of the input speech samples. Their proposed network consists of four convolutional layers followed by two fully-connected layers and is finalized by two LSTM layers. The network was jointly trained on the TIMIT dataset using a CTC loss function and grayscale images from mel-filter banks as input. Given the short duration of the individual phonemes, the system operates on audio snippets of 15 to 25 milliseconds. Similar to Shi et al., the authors apply rectangular pooling layers to obtain longer feature vector sequences [31]. The reported results compete with traditional Gaussian mixture models and hidden Markov models for ASR tasks.

Amodei et al. presented the Deep Speech 2 system for speech recognition. The system supports English and Mandarin Chinese as input languages [38]. The CRNN employs only three convolutional layers followed by seven bidirectional RNN or GRU layers. The model was trained end-to-end using a CTC loss function and a sequence of spectrograms of power-normalized audio as input. The system outputs a sequence of graphemes and uses a language model together with beam search to reconstruct words from audio. The authors found that both the LSTM and GRU cells performed similarly well as the RNN layers. GRU cells, however, needed less computations and were less likely to diverge. For the CNN part, the paper evaluated both the use of 1D, time-only domain convolutions and 2D, frequency–time domain convolutions. 2D convolutions fared better, especially with regard to noisy data. Deep Speech 2 was trained with 12 000 hours of English speech and 9000 hours of Mandarin Chinese. Additionally, it used an increased dataset with augmented data both boosting the effective corpus size as well as improving the system's noise robustness. The authors also evaluated the system with accented speech and noisy read-outs in coffee shops, streets, and so on. In both scenarios, the model's performance deteriorated.

4.3 Methods of Input Data Representation

There are many different types of audio representations used in spoken-language processing. Many higher-level characteristics of sound relate to the energies of different frequency bands. This explains the utility of time–frequency representations of audio, such as spectrograms, which are frequently used in the literature [39, 40, 41, 42, 43]. Alternatively, classical machine learning systems and i-vector systems usually rely on *mel-frequency cepstral coeffiecient* vectors (MFCC) [36, 44, 45] or derivatives thereof such as *perceptual linear prediction coefficients* (PLP) [33]. Other researchers have evaluated the use of raw waveform audio directly [46, 47].

Collobert et al. introduced Wav2Letter, an end-to-end convolutional neural network speech recognition system [47]. The authors evaluated their system with three different input representations: mel-frequency cepstral coeffiecient, spectrograms, and raw waveform data. In their paper, a fully convolutional neural network performed best with MFCC vectors as input. Yet, power spectrograms still outperformed raw waveform inputs. This observation corresponds with that of Dieleman et al., who also noted that spectrograms are computationally cheaper, given their already reduced and compacted representation [46]. Raw audio, especially when sampled at a high rate of 44 kHz, increases the amount of striding width and window sizes of the employed convolutional layers. Deng et al. reported noticeably fewer speech recognition errors using large-scale deep neural networks when using mel-scale filter bank spectrograms, compared to MFCC features [48].

4.4 Language Identification Using i-Vector Systems

Prior to the neural-network-based deep learning systems mentioned above, many researchers focused on so-called *identity vector* systems (shortly, *i-vector* systems) for spoken-language processing tasks. Dehak et al. introduced i-vector systems for speaker verification tasks [44]. i-vectors are a representation obtained by mapping a sequence of frames of a given utterance onto a low-dimensional vector space, based on a factor analysis technique. This is referred to as the *total variability space*. Typically, such a system is designed as follows. First, a feature extractor is used to turn an audio file into a 20-dimensional MFCC vector or a 56-dimensional *shifted delta cepstral* vector (SDC). Both vectors are then used to train a *unified background model* (UBM), a speaker- and language-independent *Gaussian mixture model* (GMM). Different speakers are clustured into different acoustical subspaces within the UBM. The trained, high-dimensional GMM supervector is then decomposed into its individual components to obtain respective speaker- and language-dependent characteristics. For this purpose, matrix decomposition methods such as *principal component analysis* (PCA) are employed. During decomposition, the i-vector is whitened by subtracting a global mean, scaled by the inverse square root of a global covariance matrix, and then normalized to unit length [45]. Typically, the i-vector is a compact representation of 400 to 600 dimensions. Finally, a score between the model and the i-vector of a test sample is computed. The simplest scoring measure is the cosine distance between the embedded i-vectors of the analyzed languages.

Other research experimented with *linear discriminant analysis* (LDA) and *neighborhood component analysis* (NCA) for dimensionality reduction of the UBM [44]. Sizov et al. reported using *probabilistic linear discriminant analysis* (PLDA) for this purpose [49].

While many systems share i-vectors as the input to a classifier, the actually used classification algorithms vary widely. In the context of language identification, Dehak et al. used *support vector machines* (SVM) with cosine kernels [44]. Other researchers employed logistic regression [50] or used simple, three-layer neural networks [51]. Gonzalez et al. outperformed all previous attempts by using a four-layer deep neural network [52]. Gelly et al. presented the use of bidirectional LSTMs with i-vectors [53]. Other submissions to the NIST LRE 2015 challenge [54] include many further research approaches with i-vector systems, among them using stacked bottleneck features, Bayesian unit discovery, and model fusions [55, 56, 57]. The extent of feature engineering around i-vectors results in more complex systems with an increasing number of computational steps in their pipeline. Zazo et al. did a comparison analysis between i-vector systems and LSTM networks and found the latter to perform better in some settings and to be on par in others [58]. They also note that the LSTM network used about 85 % fewer parameters than classical i-vector systems, while achieving robust and comparable results in several challenging scenarios.

4.5 Methods for Data Augmentation

Last, we present work related to *data augmentation*. To overcome insufficient varieties in datasets or to extend their sizes, researchers supplement their data with artificially created or augmented files. This approach not only boosts the overall dataset size but helps to improve a model's ability to generalize by learning from more diverse samples. For computer vision tasks, Szegedy et al. proposed to scale, translate, rotate, deform, and mirror the input images [25]. All these modifications are, however, some form of image manipulation. For audio data, there are alternative approaches, too. Similar to image scaling, one can change the playback speed of audio data by changing the sampling rate. There are two downsides to this approach, though. First, randomized time stretching modifies the audio pitch. Speeding up audio leads to unnaturally high-pitched voices. Second, manipulating the time domain alters the duration of pauses between words and frequency activations. Such extreme modifications could render the augmented data incompatible with the original dataset. Ko et al. recommended changing the speed of the audio signal to no less than 90 % and no more than 110 % of the original signal speed [59]. The largest benefit of this augmentation method are its simplicity and low implementation effort.

Amodei et al. noted that introducting background noise into their data helped to improve speech recognition robustness for noisy speech samples in general [38]. They augmented about 40 % of their dataset with randomly selected audio clips. We explore this approach in our system as well (see Section 7.4.7). Cui et al. proposed using a stochastic feature mapping in order to transfer one speaker's speech features to another speaker [60]. Languages such as Bengali and Assam are spoken by small communities only and, hence, have only a limited supply of digital available speech recordings. The proposed approach attempts to artificially extend the amount of available sound samples by applying voice conversion. Jaitly et al. introduced what they call *vocal tract length perturbation* (VTLP) to improve speech recognition systems [61]. Inspired by previous work on *vocal tract length normalization* [62], which removes speaker-to-speaker variations using normalization methods, Jaitly et al. warped the frequency of each utterance by a random factor. Using VTLP, recognition errors on the evaluation dataset could be decreased.

5 Dataset Compilation

In this chapter, we explain the structure of the datasets used throughout this thesis and how we obtained them. Furthermore, we address other datasets used in speech processing and discuss why they where not suited for our research.

Recent advances in deep learning were fueled by the availability of high-performance hardware, especially massively parallel graphics processing units (GPUs), and of large-scale, well-annotated, public datasets, for example ImageNet in the computer vision domain [22]. Historically, within the language identification community, the TIMIT corpus of read speech has long been the default test set [10]. TIMIT contains a total of 5.4 hours of speech, consisting of ten sentences spoken by 630 speakers from eight major dialect regions of the United States. All samples are recorded at 16 kHz, which is a significantly lower quality than 44.1 kHz and 48 kHz, which are the standards today. Given the short span of each individual sound clip, the overall corpus duration, and restriction to only one language, TIMIT was unsuited for this thesis. Therefore, it was necessary to obtain our data elsewhere.

In this thesis, two primary datasets were collected. We process speeches, press conferences, and statements from the European Parliament, as well as news broadcasts sourced from YouTube.

5.1 Language Selection

For the scope of this thesis, we decided to limit ourselves to a number of languages spoken by many millions around the world. We focus our efforts on languages with many publicly available speeches from the various data sources explained below, namely the EU Speech Repository and YouTube. From a linguistic standpoint, we also made sure to include languages from within the same language families with similar phonetics for comparison reasons (more on the similarities of related languages in Section 7.4.5). Following these guidelines, we decided on two Germanic languages, English and German, and two Romance languages, French and Spanish. We later extended our selection to Russian and Mandarin Chinese.

5.2 EU Speech Repository

The EU Speech Repository[3] is a collection of video resources for interpretation students provided for free by the European Commission. The dataset consists of debates of the

[3] `https://webgate.ec.europa.eu/sr/`, accessed 10 March 2017

European Parliament as well as committee press conferences, interviews, and dedicated training materials from EU interpreters. Each audio clip is recorded in the speaker's native language and features exactly one speaker. Overall, however, the dataset consists of many different male and female speakers, adding a nice variety to the data.

With 131 hours of speech data, this dataset is significantly smaller than the YouTube dataset (see Section 5.3). We obtained material in four languages: English, German, French, and Spanish. Prior to downloading the data, we gathered and processed every web page containing a single video in our target language by using the Selenium website end-to-end testing framework. We downloaded and extracted the audio channel of the source videos using the command line tool `youtube-dl`.[4]

5.3 YouTube News Collection

Following the approach of Montavon, who used podcasts and radio broadcasts as input data [39], we looked for large, public sources of speech audio. Both podcasts and radio stations have disadvantages for the language identification task of this thesis for several reasons. Podcasts are usually restricted to one single speaker and lack variety. Radio, on the other hand, contains much less speech content and consists mainly of music and songs. Consequently, we decided on using news broadcasts, which provide high-quality male and female speech audio data suitable to our needs. To obtain a large variety of languages and gather enough hours of speech audio, we sourced the majority of our data from YouTube.

For each target language, we manually selected one or more YouTube channels of respected news outlets, for instance, BBC and CNN for the English language. Using more than one channel for each language has the benefit of collecting a wide variety of accents, speech patterns, and intonations. For a full list of channels, refer to Table 1. All channels were chosen regardless of their content, their political views, or journalistic agenda. Again, we downloaded the data using the command line tool `youtube-dl` and saved only the audio channel.

Audio obtained from news coverage has many desired properties. The data is of high recording quality and hundreds of hours of recordings are available online. News anchors are trained to speak naturally and conversationally, while maintaining a steady speed of about 150 words per minute [63]. News programs often feature guests or remote correspondents resulting in a good mix of different speakers. Unlike audio book recordings with texts read aloud, news anchors converse in a regular, human, conversational tone with each other. Last, news programs feature all the noise one would expect from a real-world situation: music jingles, nonspeech audio from video clips and transitions between reports.

[4] `https://github.com/rg3/youtube-dl`, accessed 23 March 2017

YouTube Channel Name	**Language**
CNN	English
BBCNews	English
VOAvideo	English
DeutscheWelle	German
Euronewsde	German
N24de	German
France24	French
Antena3noticias	Spanish
RTVE	Spanish
VOAChina	Mandarin Chinese
Russia24TV	Russian
RTrussian	Russian

Table 1: YouTube channel names used for obtaining the speech data and their corresponding languages.

	EU Speech Repository	**YouTube News**	**YouTube News Extended**
Languages	English, German, French, Spanish	English, German, French, Spanish	English, German, French, Spanish, Russian, Chinese
Total Audio Duration	131 h	942 h	1508 h
Average Clip Duration	7 min 54 s	3 min 18 s	4 min 22 s
Audio Sampling Rate	48 kHz	48 kHz	48 kHz

Table 2: Comparison of our collected EU Speech Repository data and the YouTube News dataset. With about 1000 hours of audio recordings, the acquired YouTube dataset is ten times larger than the EU Speech Repository.

On the one hand, this might improve the model's noise robustness. On the other hand, this might interfere with training, for example, when having audio segments containing very low speech-to-duration ratios. Although some broadcasts feature mixed language parts, such as the names of foreign cities, companies, and persons, we believe this is not a big problem. The pronunciation and intonation of these words and phrases still follow the host's native language.[5] In essence, we believe that speech data sourced from news broadcasts represents an accurate, real-world sample for speech audio.

In contrast to our EU Speech Repository, this dataset consists of about 1000 hours of audio for the same four languages: English, German, French, and Spanish. Additionally, we also gathered an extended language set adding Mandarin Chinese and Russian. The extended set is only used for evaluating the model extensibility, as outlined later (see Section 7.4.8). Table 2 provides a comparison between the two datasets.

5.4 Other Datasets

The decision to source our own dataset was largely influenced by the limited availability, lack of language choices, and the number of samples of existing speech datasets. The

[5] As an example, according to the international phonetic alphabet (IPA), the city of Berlin has many different pronunciations in different languages: [bɛʁˈɩiːn] (German), /bə.ˈlɪn/ (British English), and [bɚ.ˈlɪn] (American English).

Linguistic Data Consortium (LDC)[6] collects and maintains many speech datasets. Unfortunately, access to their dataset collection is restricted to paying members only. For instance, the aforementioned TIMIT corpus is amongst their datasets. The LDC also maintains the NIST Language Recognition Evaluation challenge dataset (LRE) [54].

Other datasets such as the Wall Street Journal corpus, a collection of read-outs from the WSJ, is only available in English and, hence, does not fit our research task of identifying multiple languages [64]. Similarly, the Libri Speech corpus is restricted to 1000 hours of read-aloud English speech as well [65]. This data is derived from free public domain audio book readings from LibriVox,[7] which also covers other languages than English. However, as stated earlier, we deliberately decided against using read-out speech, as we found it unrepresentative of regular, conversational speech. Another free public domain dataset is Voxforge,[8] which includes speech samples in different languages. Yet, for our deep learning approach, we found the number of available samples per language lacking. Their English data covers less than ten hours of audio data.

Finally, some large-scale datasets remain unpublished. Gonzalez-Dominguez et al. collected and documented the Google 5M LID corpus, a collection of voice recordings from various Google services including voice search and the speech input API on the Android operating system [52, 66]. Similarly, Baidu built their own proprietary Mandarin Chinese corpus consisting of 9400 hours of speech data for their DeepSpeech 2 system [38].

[6] `https://www.ldc.upenn.edu`, accessed 15 May 2017
[7] `https://librivox.org/`, accessed 15 May 2017
[8] `http://www.voxforge.org`, accessed 15 May 2017

6 Implementation

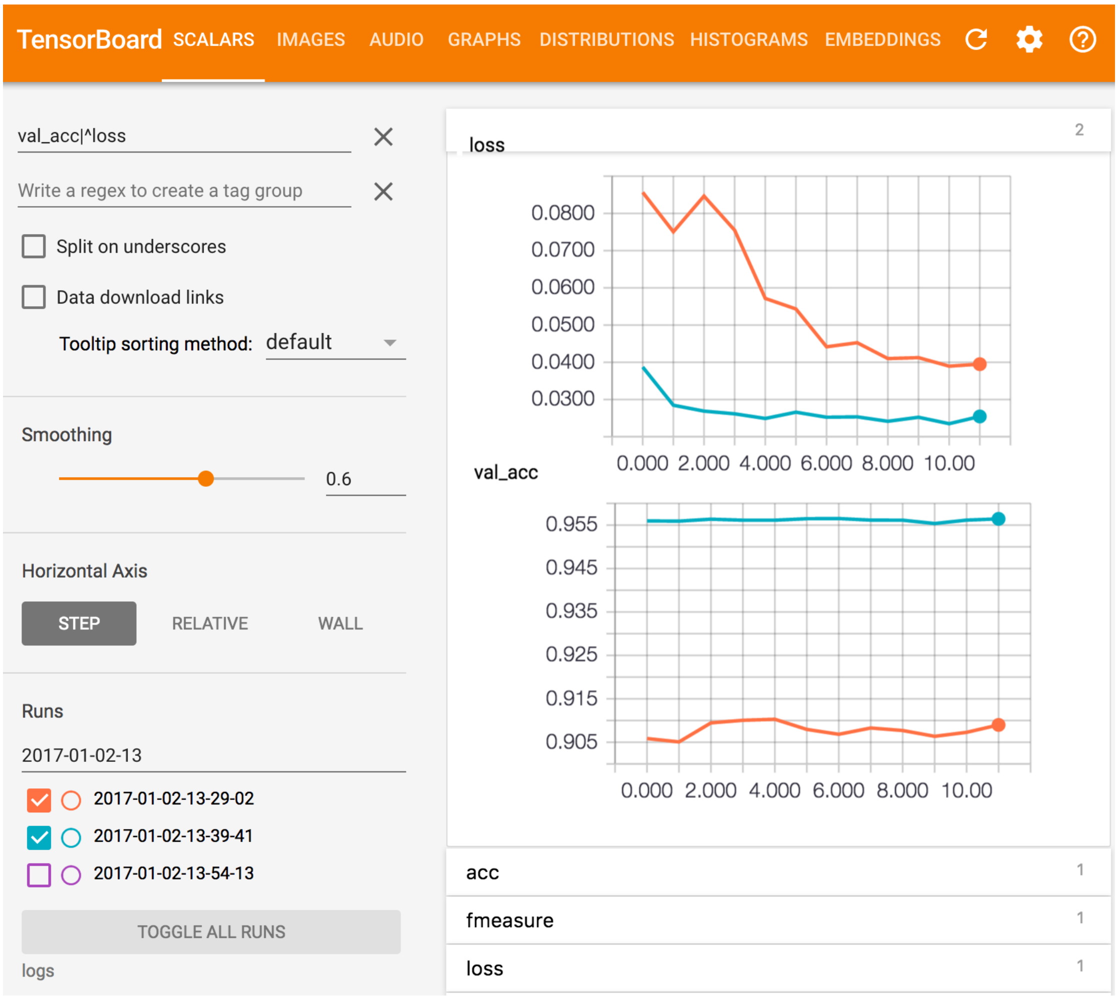
This section outlines the software resources and frameworks used for implementing our language identification system. Given the datasets presented in Chapter 5, we describe the necessary data preprocessing steps to obtain the input audio representations for our network, as explained in Chapter 3. Finally, we describe the concrete layer-wise model architecture of our CNNs and CRNN in detail.

6.1 Software

Our language identification system is implemented in Python 3 and uses the open-source deep learning framework Keras [67] with the TensorFlow backend [68] for training our neural networks. Keras provides us with a set of higher-level machine learning primitives (such as convolutional layers) and optimization algorithms (including *stochastic gradient descent*), without sacrificing any fine-grained control over the training parameters. Internally, Keras builds on Google's open-source numerical operation library TensorFlow, which is optimized for quickly computing multidimensional data on GPUs. We make heavy use Keras's aforementioned primitives, especially convolutional layers and the efficient LSTM implementation. Using the TensorFlow framework not only future-proofs our implementation but also enables us to readily deploy our model on mobile phones in the future.

All models are stored on disk during training, including a summary of the layer architecture as well as the models' weights. This makes it easy to load, evaluate, and deploy all the models later. During the evaluation phase, all performance metrics (accuracy, precision, recall, F1 score) are calculated using the Scikit Learn framework [69]. All measurements are logged and visualized using TensorBoard,[9] both for the training and the validation set. Having the ability to easily study and compare different metrics such as accuracy and the loss value across several training runs made it very comfortable to continuously monitor the progress of our research. Figure 7 shows a TensorBoard instance with plots of the metrics for several models. We regularly compared different approaches and models against each other and used these plots to select the best performing systems for further experiments.

[9] `https://www.tensorflow.org/how_tos/summaries_and_tensorboard/`, accessed 30 January 2017



val_acc|^loss

Write a regex to create a tag group

□ Split on underscores

□ Data download links

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

2017-01-02-13

☑ ○ 2017-01-02-13-29-02

☑ ○ 2017-01-02-13-39-41

□ ○ 2017-01-02-13-54-13

Figure 7: A TensorBoard instance showing training loss and validation accuracy of two different models. We regularly tried different approaches and model architectures and plotted multiple evaluation measures for comparison. This workflow enabled us to judge the impact of each experiment and measure the overall performance progress of the system.

6.2 Data Preprocessing

All audio files undergo preprocessing before being fed to the neural network. As a first step, all files are encoded in the uncompressed, lossless Waveform Audio File Format (WAVE[10]), commonly known by its file extension `*.wav`. This conversion has two advantages. First, a lossless data codec allows for future audio manipulations without any deterioration in signal quality and, second, makes the data easily interchangeable with third-party programs and libraries such as SciPy [70].

Since none our neural networks operate on raw waveform audio signals directly, we transfer our features into the image domain. As introduced in Section **??**, we use a spectrogram representation of the audio data for training our models. The spectrograms are generated using the open-source command line tool SoX.[11] The spectrograms are discretized using a Hann window [71] and 129 frequency bins along the frequency axis (y axis), as instructed by the SoX manual.[12]

Male human voice frequencies begin between 150 Hz and 255 Hz [72]. Female voice frequencies are usually shifted by one octave and begin at 210 Hz. Typically, voice frequencies range up to 3.4 kHz. For reference, the human ear is capable of recognizing frequencies from 20 Hz to 20 kHz, with most sensitivity in the region of between 300 Hz and 10 kHz. Single sounds, however, exceed this limit significantly. Generally, voice frequencies are influenced by gender, age, as well as various other factors, such as the language, the type of discourse, and the emotional state of the speaker. Figure 8 shows the frequency areas with high energy in response to the tone of different vowels of human speech for the English language.

Most phonemes in the English language do not exceed 3000 Hz in conversational speech. Consequently, we instructed SoX to only include frequencies of up to 5 kHz in the spectrograms. The time axis (x axis) is rendered at 25 pixels per second. Each audio sequence is clipped into nonoverlapping ten-second segments. To avoid padding issues and segments shorter than ten seconds, the final segment is discarded. Since we gathered enough training data, we decided against padding with black pixels, which could be interpreted as silence and add unnaturally long speech pauses. We also decided against filtering silent sections within the ten-second audio segments to preserve the natural pauses between words and to not disturb the regular speech rhythm.

Frequency intensities are mapped to an eight-bit grayscale range. We combined all audio channels into a single mono channel in order to generate only a single spectrogram image. The resulting grayscale images are saved as lossless, 500×129 PNG files. Listing 1 shows the SoX command for generating a spectrogram image from an input audio file.

[10] `http://www.microsoft.com/whdc/device/audio/multichaud.mspx`, accessed 23 February 2017

[11] `http://sox.sourceforge.net/`, accessed 23 February 2017

[12] `http://sox.sourceforge.net/sox.pdf`, p. 32, accessed 26 March 2017

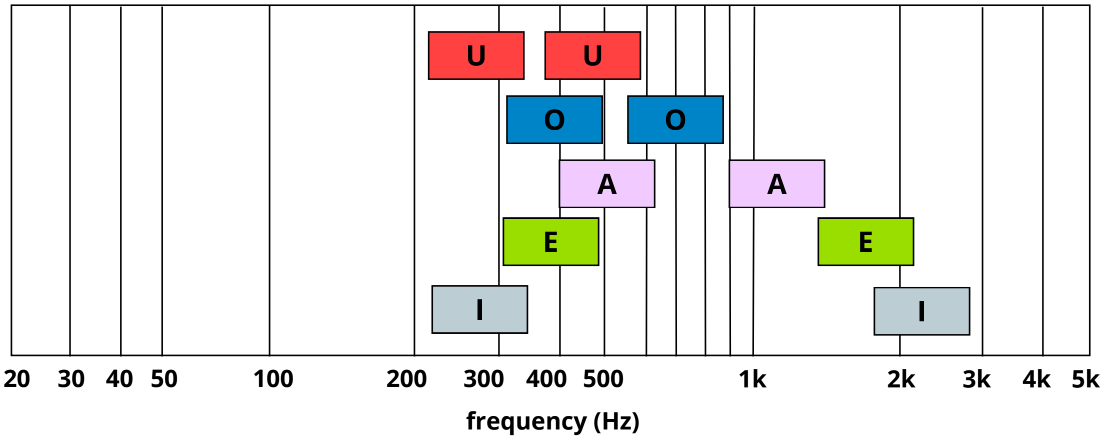


Figure 8: The upper and lower frequency areas (*formant*) for different vowels, typical for human speech. The lower speech formant F_1 has a total range of about 300 Hz to 750 Hz and the higher speech formant F_2 about 900 Hz up to over 3000 Hz. But each single spoken tone has a much narrower range for both formants.

```
sox -V0 input.wav -n remix 1 rate 10k spectrogram -y 129 -X 50 -m -r -o spectrogram.png

V0 - verbosity level
n - apply filter/effect
remix - select audio channels
rate - limit sampling rate to 10k; caps max frequency at 5kHz according to Nyquist-Shannon sampling theorem
y - spectogram height
X - pixels per second for width
m - monochrome output
r - disable legend
o - output file
```

Listing 1: SoX command and options used for generating monochrome spectrograms. All audio files were discretized into 129 frequency buckets using a constant pixel width per time step, resulting in spectrogram images of 500 × 129 pixels.

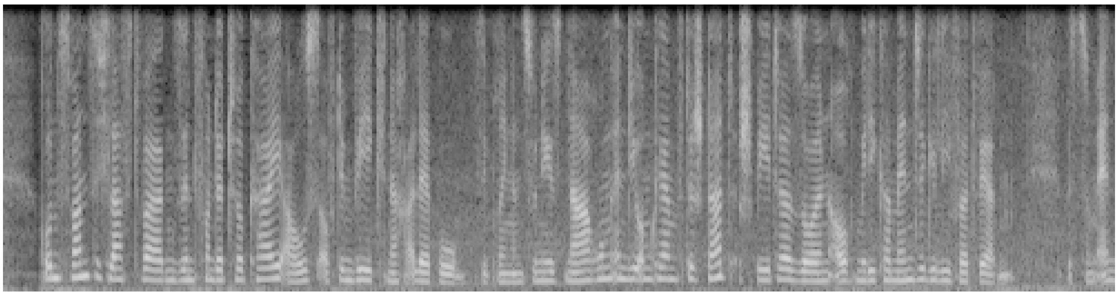


Figure 9: A spectrogram generated from a German ten-second audio clip using SoX. Notice the bright ripple-like patterns representing intense frequency activations. We hypothesize that these frequency activations are suitable as the main features for the classifier.

As seen in Figure 9, the spectrograms feature very apparent bright ripple-like patterns. Each of these patterns represents a strong activation of a certain frequency at a point in time. Several frequency activations can be active simultaneously, constituting a particular phoneme or sound. A sequence of these phonemes forms words and is only interrupted by short speech pauses. We hypothesize that our LID system learns the characteristical and unique composition of these frequency activations for every language in our classifier.

6.3 Neural Network Architectures

To successfully solve our research task with deep neural networks, we have to design a fitting model architecture. Combing the right layers and adjusting the parameters correctly is not a trivial task. Critics argue that deep learning systems are a black box and that the impact of the individual layout of the network layers is hard to understand. Therefore, we base our model architectures on proven existing designs from related work and adapt them to our needs, while heeding best practices [73, 27].

The architecture and the number of parameters of a deep neural network suited for a given task is determined by the bias–variance tradeoff [74]. It is the goal of a network's author to avoid underfitting or overfitting the resulting model. Bias characterizes the prediction error that results from the model's limited ability to learn relevant relations between features in the training data. Underfitting can be caused by designing too small or too few network layers, causing a high bias. Variance, in contrast, refers to the error that results from the model responding overly sensitive to variation in the training data. Designing a too large network, introducing too many parameters, or adding too many layers result in a model with low variance. Hence, the model learns an exact representation of the training data without abstracting for more general applications. This is known as *overfitting*.

Designing and adjusting the network layout is an iterative process. We tried many variations and parameters of our proposed model layouts to find the most suitable design for our LID system. The layout of the network interdepends and interacts with other design decisions of the system, especially the loss calculation. Each change in parameters involves a complete new training run to judge the effect of the change. Hence, the architecture always needs to be tested in its entirety.

In this thesis, we tried three different designs of convolutional neural networks. The first and second ones are based on the early VGGNet-like CNN-M model architectures by Simonyan et al. [26]. The CNN features five convolutional layers and modest feature map output sizes. Note that the network's first two convolutional layers have comparatively large kernel size of 7 × 7 pixels and 5 × 5 pixels, respectively, yielding a large receptive field. Each convolutional layer is followed by batch normalization [30], a technique that helps in increasing training speed and achieving a higher level of model regularization. Following these layers, we added a 2 × 2 pooling layer with a stride of 2. After the five convolutional layers, we added regularization through a 50 % dropout before flattening all parameters to a fully-connected layer with 1024 output units. The final, fully-connected layer serves as a classifier that outputs the language identification predictions. Henceforth, we refer to this model as CNN_A. The full network layout can be seen in Table 3.

A slightly adapted version of CNN_A has the same number of convolutional layers but features fewer feature maps. Instead of doubling the initial value of 16 feature maps for every convolutional layer, we stick to a schedule of 16–32–32–64–64 feature maps, respectively. The fully-connected layer is also reduced to only 256 output units. Overall, this model has significantly less parameters than CNN_A. The purpose of this variation is to ensure that the architecture proposed for CNN_A is not unnecessarily complex. We called this variation CNN_B. The network layout of CNN_B is also listed in Table 3.

Lastly, we evaluate architecture CNN_C, which is based on the VGGNet-16 network [24] and uses constant kernel size of 3 × 3 for all convolutional layers. At the same time, we increase the number of convolutional layers to seven and extend the number of feature map outputs for each layer to 64–128–256–256–512–512–512. In CNN_C, all fully-connected layers of are identical to CNN_A. The main difference of this network design are smaller kernel sizes and, hence, a smaller receptive field of the convolutional layers. To compensate for this, we increase the number of layers. The complete CNN_C architecture is laid out in Table 4.

For our CRNN hybrid network, we construct a convolutional neural network followed by a recurrent neural network. Specifically, we use a bidirectional *long short-term memory network* (LSTM) for the RNN part. We decided on using a bidirectional model of two LSTMs instead of a single LSTM based on the successful results of Shi et al. [31]. The CNN part of this network is tasked with extracting visual features and providing a high-dimensional intermediate frequency representation. The RNN part splits the intermediate interpretation into a series of vectors (along the x axis), which it treats as if they belonged

Layer Type	**Output Size**		**Kernel**	**Stride**
	CNN_A	**CNN_B**		
Convolution	16	16	7×7	1
Max Pooling	16	16	2×2	2
Convolution	32	32	5×5	1
Max Pooling	32	32	2×2	2
Convolution	64	32	3×3	1
Max Pooling	64	32	2×2	2
Convolution	128	64	3×3	1
Max Pooling	128	64	2×2	2
Convolution	256	64	3×3	1
Max Pooling	256	64	2×2	2
Dropout & Flatten	3328	832		
Fully-Connected	1024	256		
Fully-Connected	4	4		

Table 3: The layerwise architecture for the convolutional neural networks CNN_A and CNN_B. These designs are based on early VGG-like networks and features large kernel size for the first two convolutional layers in an effort to capture a large receptive field of features.

Layer Type	Output Size	Kernel	Stride
Convolution	64	3×3	1×1
Max Pooling	64	2×2	2×2
Convolution	128	3×3	1×1
Max Pooling	128	2×2	2×2
Convolution	256	3×3	1×1
Convolution	256	3×3	1×1
Max Pooling	256	2×2	2×2
Convolution	512	3×3	1×1
Convolution	512	3×3	1×1
Max Pooling	512	2×2	2×2
Convolution	512	3×3	1×1
Max Pooling	512	2×2	2×2
Flatten	6144		
Fully-Connected	1024		
Fully-Connected	4		

Table 4: The layerwise architecture for the convolutional neural network CNN_C. With seven convolutional layers, this network design is deeper than the other two proposed CNN architectures. Additionally, the number of feature maps is increased to 512 units compared to the 256 units of the CNN_A network. Overall, this network consists of a higher number of parameters than present in the other two designs.

Layer Type	Output Size	Kernel	Stride
Convolution	$123 \times 494 \times 16$	7×7	1×1
Max Pooling	$61 \times 247 \times 16$	2×2	2×2
Convolution	$57 \times 243 \times 32$	5×5	1×1
Max Pooling	$28 \times 121 \times 32$	2×2	2×2
Convolution	$26 \times 119 \times 64$	3×3	1×1
Max Pooling	$13 \times 59 \times 64$	2×2	2×2
Convolution	$11 \times 57 \times 128$	3×3	1×1
Max Pooling	$5 \times 56 \times 128$	2×2	2×1
Convolution	$3 \times 54 \times 256$	3×3	1×1
Max Pooling	$1 \times 53 \times 256$	2×2	2×1
Transpose	$53 \times 1 \times 256$		
Reshape	53×256		
Bidirectional LSTM	1024		
Fully-Connected	4		

Table 5: The layerwise architecture of the convolutional recurrent neural network. The network consists of two parts, a CNN and a bidirectional LSTM. This design shares its CNN architecture with the previously introduced CNN_A. The final convolutional layer is sliced into time steps along the x axis (time axis) and serves as input to LSTM.

to consecutive time steps.

In the CNN part, we repurpose the CNN_A network architecture of five convolutional layers with larger kernels for the first two layers, since we found this CNN layout to work best for our LID task (details are provided later in Section 7.4.4). We also keep the batch normalization and max pooling layers. The bidirectional LSTM layer consist of two single LSTMs with 256 output units each. We concatenate both outputs to a vector of 512 dimensions and feed this into a fully-connected layer with 1024 output units serving as the classifier. The complete model architecture can be seen in Table 5.

double check output sizes

We decided on training both parts of the hybrid network separately. First, we trained the CNN part by itself. This simplifies the task of learning the frequency features for the network. In practical terms, this means that we could reuse the model weights of the previously trained CNN_A model. For the RNN part, we used a fine-tuning approach. This means that we froze all convolutional layers' weights by disallowing any further weight updates during the training phase. The bidirectional LSTM and FC layers, however, remained unfrozen and continued to be subject to weight updates and were, hence, trained regularly. Later, we also experimented with unfrozen RNN weights, meaning that we trained and updated both the CNN and LSTM layers at the same time. This approach, however, yielded worse results, and we did not pursue it further.

	Machine A	Machine B
OS	Ubuntu Linux 14.04	Ubuntu Linux 16.04
CPU	Intel Core i7-4790K at 4.0 GHz	AMD FX-8370 at 4.0 GHz
RAM	16 GB	32 GB
GPU	Nvidia GeForce GTX 980	Nvidia Titan X
VRAM	4 GB	12 GB
Images/s		

Table 6: Hardware resources used in training the neural networks. We made heavy use of modern GPUs to benefit from hardware-accelerated numerical computations.

7 Experiments and Evaluation

In Chapter 6, we detailed our network architecture and its implementation. In this chapter, we study how our network performs given the datasets presented in Chapter 5. We present and discuss the results of training the outlined neural network architecture for spoken-language identification. We perform an evaluation on our system, while assessing several performance metrics.

Further, we experiment with modified model architectures to maximize the model's accuracy, improve its noise robustness and study the effect of background music on the neural network. To assess the real-world performance of the LID system, we augment our data to simulate various noisy environments. Lastly, we evaluate the classification performance of our approach and discuss the system's interlanguage discrimination capabilities and extensibility to other languages.

7.1 Hardware Resources

In order to facilitate Keras's and TensorFlow's hardware-accelerated computation, we executed all training runs on CUDA-compatible[13] GPU machines at our disposal at the Internet Technologies and Systems chair of the Hasso Plattner Institute. Details are shown in Table 6.

measure images/sec

[13] https://developer.nvidia.com/cuda-zone, accessed 30 January 2017

	European Speech Repository	**YouTube News**
Training Set	18 788	193 432
Validation Set	5372	55 272
Test Set	2684	27 632
Total	26 844	276 336

Table 7: The number of samples for our training (70 %), validation (20 %), and testing (10 %) sets taken from the two studied datasets.

7.2 Data

For our performance evaluation, we use the European Speech Repository and YouTube News dataset, as described in Chapter 5. Both datasets were preprocessed and converted to spectrogram images (see Section 6.2). Each spectrogram image represents a nonoverlapping ten-second snippet of a source audio file. We chose this duration in reference to the NIST LRE2015 challenge [54]. We split both datasets into a training (70 %), a validation 20 %, and a testing set 10 %, and all files were distributed equally between all four language classes. The number of samples per class was limited by the language with the least number of files to ensure an equal distribution across the classes. The European Speech repository yields a total of about 19 000 training images, which amounts to roughly 53 hours of speech audio. The YouTube News dataset is considerably larger and yields a total of about 194 000 training files, or 540 hours. Table 7 contains the detailed dataset splits.

Given the European Speech Repository's smaller size, we only used it initially to confirm the validity of our approach. At the point at which we were satisfied with the results, we did not include it in the extensive robustness tests that we used for the evaluation on the YouTube News dataset. Moving on to a bigger challenge, we augmented the original audio of the YouTube News dataset with three different background noises to evaluate how well our model performs in nonideal, real-world situations outside of a news broadcasting studio. For the first experiment, we added generic white noise to the data. For the second experiment, we added noise to simulate an old phone line or bad internet connection during a voice chat. In a last experiment, we added background music to the data. All experiments are described in detail below.

7.3 Training the Neural Network Model

Neural networks have a multitude of hyperparameters that influence the training results drastically. In this section, we briefly explain our choice of hyperparameters alongside other important training settings.

Optimizer We use *adaptive moment estimation* (Adam) [75] as our optimization method to quickly and efficiently reach convergence of our model. Adam utilizes momentum during gradient updates to support a quicker convergence. We set the optimizer's parameters β_1 to 0.9, β_2 to 0.999, and ϵ to 10^{-8}. For the majority of trainings, we use Adam and only resort to plain stochastic gradient descent during fine-tuning when we need more control over the learning rate schedule and want smaller weight updates.

Weights Initializer All layer weights are initialized within the range $[0, 1]$ using a normalized *Glorot uniform initializer* (also known as *Xavier initializer*) [76]. This heuristic is designed as a compromise between the two goals of initializing all layers with the same activation or gradient variance, respectively. Thus, the biases are initialized with 0 and the weights $\boldsymbol{W}$ of each layer with the following heuristic [16, p. 303]:

$$\boldsymbol{W} \sim U\left(-\frac{6}{\sqrt{n_j + n_{j+1}}}, \frac{6}{\sqrt{n_j + n_{j+1}}}\right)$$

where $U(-a, a)$ is the uniform distribution in the interval $[-a, a]$ and n_j and n_{j+1} are the sizes of the previous and current layer, respectively.

Data Normalization The grayscale images are loaded using SciPy and normalized to values in the range of $[0, 1]$. The shape of all inputs needs to be uniform across all samples and is set to $500 \times 129 \times 1$, unless otherwise noted. Data loading is handled through Python generators to keep the system's memory footprint low.[14]

Learning Rate We set the initial learning rate to 0.001. Given the Adam optimizer's dynamic adaptation of the learning rate, we expect the learning rate to be automatically increased or decreased after every iteration. Therefore, we do not specify a manual learning rate decay schedule.

Batch Size We specify the batch size depending on the available VRAM of the training machine. We use 64 images per batch for Machine A and 128 images per batch for Machine B (see Section 7.1 for the hardware specifications).

Weight Regularization comprises any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. We

[14] `https://docs.python.org/3/glossary.html#term-generator`, accessed 30 January 2017

employ the L^2 norm [16, p. 231] (sometimes also known as *weight decay*) as a weight regularizer for all convolutional and fully-connected layers to improve the model's generality. This regularization strategy drives the weights closer to the origin by adding a regularization term to our loss function. We penalize the loss value with a weight decay value of 0.001. Additional regularization happens through the use of batch normalization layers.

Epochs We limit the model training phase to a maximum of 50 epochs when using the Adam solver. We usually reach convergence well below this threshold. For training sessions with SGD, we increas this parameter considerably, since the weight updates per epoch are smaller, and more training iterations are needed to reach convergence. To speed up our workflow, we employ an early-stopping policy and stop training if the validation accuracy and loss value do not considerably change within a ten-epoch window.

Metrics We observe loss, accuracy, recall, precision, and F1 score for both the training and validation set while training the model. All values were are to log files and visualized as graphs in TensorBoard. A complete summary of all assessed metrics can be found in Section 7.4.1.

Loss As is common for multivariate classifications, all models are trained with a softmax cross-entropy loss function.

7.4 Evaluation

7.4.1 Evaluation Metrics

In this section, we discuss the evaluation metrics used throughout our experiments. All metrics are generally defined for binary classification scenarios only. Given our multiclass classification problem, we report the average of the individual class performance measures in the following sections.

Accuracy is a common measure in machine learning and is defined as the ratio of correctly classified samples to all samples in the dataset. In the context of language identification, this translates to:

$$\text{accuracy} = \frac{|\{\text{correctly identified language samples}\}|}{|\{\text{all language samples}\}|}$$

Precision and Recall *Precision* defines the proportion of retrieved language samples that are correctly identified as belonging to said language. *Recall* is the fraction of correctly identified language samples to all samples belonging to this language. Both measures are calculated using true and false positives as well as false negatives. These are defined as follows. *True positives* are the samples that are correctly attributed to a given language. In contrast, *false positives* are attributed to a given language, but they actually do not belong to it. Lastly, *false negatives* are, by mistake, not recognized as part of a given language.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

The F1 Score is the scaled harmonic mean of precision and recall. It is a combined judgement of both values, since it is generally not interesting to assess one without the other.

$$\text{F1} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

7.4.2 Results for the EU Speech Repository Dataset

In order to verify our theoretic model of using convolutional neural networks for classifying audio data in the image domain, we established a baseline with the smaller EU Speech Repository dataset. Previous work with CNNs showed that a careful arrangement of the neural network layers has great effect on the final classification performance. If one does not use enough or sufficiently large layers, the model is not able to properly distinguish between classes. Going too deep or using too large layer outputs increases the overall number of model parameters to a degree where both training time and classification performance suffer again. The goal of this experiment is to find favorable settings for the number of necessary convolutional layers, the kernel sizes of the convolutions, the numbers of output maps of the convolutional layers, and finally, the number of hidden units of the fully-connected layer.

In Section 6.3, we introduced three proposals for different CNN architectures. CNN_A features large kernel sizes in the first two layers, and we expect this architecture to capture more information up-front. A slightly smaller version of this design with less parameters is CNN_B. It serves as an assertion to verify that the number of output feature maps of CNN_A is properly sized and not too large. Lastly, CNN_C features equally-sized kernels

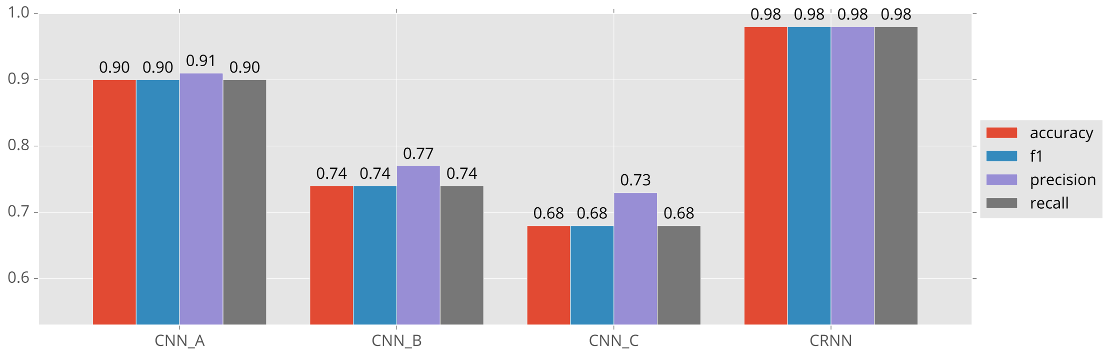


Figure 10: Performance measure comparison of three different CNN architectures and our proposal of a CRNN model evaluated on the EU Speech Repository dataset. CNN_A outperforms all other CNN approaches with a top accuracy of 0.90, but is bested by 0.98 accuracy with CRNN, showing the potential of the approach developed in this thesis.

and boasts both more convolutional layers as well as an increased number of feature map outputs.

CNN_A outperforms both of the other network architectures with respect to all the evaluated performance measures, as shown in Figure 10. With a top-1 accuracy of 0.90, CNN_A trumps CNN_B and CNN_C with 0.74 and 0.68, respectively. Comparing the F1 score, we get a very similar result: 0.90 versus 0.74 and 0.68. This experiment confirms a few of our initial hypotheses (as outlined in Section 2.2). First, convolutional neural networks are a successful technique for classifying speech audio data. Second, spectrogram images are meaningful representations for audio and retain enough information for language identification. Third, larger kernels for the initial convolutional layers are, indeed, favorable, since the increased receptive field captures both the time and the frequency domain better.

Based on these findings, we did further experimented with CNN_A. Mishkin et al. were able to gain a small improvement in accuracy by exchanging the convolutional layers' activation functions [73]. For this purpose, we replaced our ReLU activations with *exponential linear units* (ELU) [77] but were unable to measure any improvement (with only 0.85 accuracy). Shi et al. proposed to use 1 × 2 rectangular pooling windows instead of the conventional square ones [31]. This tweak yields feature maps with larger width and, hence, more features along the time domain. In theory, this should increase the CNN's sensitivity to occurrences of frequencies at certain time intervals. For this experiment, we were unable to improve (0.81 accuracy), but we come back to this technique for our CRNN approach in Section 7.4.3.

The task of this thesis is to evaluate the use of deep convolutional recurrent networks for language identification. Therefore, we extend our previously best-performing CNN_A with a bidirectional LSTM layer to capture time steps both forward from the start and backward from then end of an audio segment. Our model interprets the CNN output as an intermediate high-dimensional representation of the audio frequencies. Every vector entry along the x axis is used as a single time step and input to the LSTMs, as explained in Section 3.4. Confident in the features captured by our convolutional layers, we freeze them during the CRNN training phase to disable any further weight updates for these layers. Instead, we focus on training only the LSTMs weights and learning the frequency sequences of the audio samples. Our bidirectional LSTM layer trains two individual LSTMs with 512 outputs each, one moving forward along the input sequence and one backward. Both outputs are concatenated to form a single output vector with a dimension of 1024, which is followed by a single, fully-connected layer for classification.

Our CRNN architecture outperformed all CNN approaches significantly. A top-1 accuracy of 0.98 and an F1 score of 0.98 prove the viability of the CRNN approach and reaffirm the central hypothesis of this thesis.

7.4.3 Effect of Audio Duration

In all previous experiments, we split the audio recordings into ten-second segments, which translates to an image dimension of 500 × 129 pixels in the spectrogram. We decided on ten-second audio snippets to replicate the setup of the NIST LRE 2015 challenge [54]. To study the effect of audio duration on classification performance, we set up two versions of the EU Speech Repository dataset with nonoverlapping five- and 20-second snippets but left the number of frequency bins unchanged. Hence, we change the input dimensions to 250 × 129 pixels and 1000 × 129 pixels, respectively, and cannot just reuse our previously trained models but have to train new models.

As a baseline, we use the same CNN_A architecture as explained in Section 6.3. When completely training the five-second version from scratch, we achieve an accuracy of 0.81, falling short of the results achieved with the ten-second snippets. Next, we apply some transfer learning and fine-tune CNN_A on the five-second dataset. Since the convolutional layers are not bound to a specific input size and given that the frequency features do not change in dimension, we are able to reuse all the convolutional weights. For fine-tuning, we freeze the convolutional layers, confident in their ability to detect frequency features, and only retrain the final two fully-connected layers. After bisecting the input data, the amount of model parameters is greatly reduced, especially the fully-connected layer weights. To account for this, we fine-tune one model with a fully-connected layer of 512 outputs and a second one with the default 1024 outputs. Overall, this yields an accuracy of 0.88 and 0.89, respectively. We conclude that the effect of the smaller fully-connected layer is only marginally better.

After establishing a solid CNN foundation, we apply the same CRNN approach as previously highlighted. Due to the shorter audio duration, the final pooling layer only features five output units along the x axis, compared to the 13 output units of the ten-second CRNN. When interpreted as a sequence of time steps and fed into the bidirectional LSTM, the accuracy improves only marginally to 0.90. We suspect that the number of time steps is too little to take full advantage of the recurrent network layer.

In an effort to increase the number of output units of the final pooling layer and, hence, increase the sequence length, we apply the 1×2 rectangular pooling tweak again. We change the final two pooling layers and increase the number of output units along the x axis to 22. The y axis remains unaffected. The resulting accuracy of 0.90 and the F1 score of 0.91 remain comparable to the previous model and do not achieve improvements.

For the twenty-second version, we train the network from scratch using the CNN_A architecture. This yields an accuracy of 0.90, which is comparable to the ten-second counterpart. This experiment uses double the amount of pixels along the x axis as the ten-second baseline. This expansion of data leads to an increased computation time. Additionally, we feel that our LID system is less flexible when using longer input audio snippets. We conclude that these longer snippets do neither improve the accuracy nor make the LID system more attractive overall.

In summary, we believe that both decreasing and increasing the duration of the audio snippets used for training has a negative effect on the classification performance. While it is possible to train and fine-tune CNNs matching their ten-second counterparts accuracy-wise, we find that the CRNN approach does not boost the model effectiveness in a similar manner. Table 8 shows the results of our experiments with varying sample durations, confirming that the 10-second CRNN performed best in our evaluation.

7.4.4 Results for the YouTube News Dataset

Following the promising results from the experiments with the EU Speech Repository, we switch to the ten-times larger YouTube News dataset for further training and evaluation.

For our first experiment, we use the same CNN_A architecture as before but initialize the model weights with the weights of the best-performing model of the EU Speech Repository evaluation. Our reasoning here is to reuse the convolutional layers that are already trained to identify frequency features. However, with an accuracy of only 0.79, CNN_A does not perform as well as anticipated. One reason for this could be that the EU dataset is a lot smaller and does not feature as many diverse situations as present in news broadcasts. In the case of broadcasts, all audio is recorded in a similar environment without much background noise and exhibits high signal quality.

Model Architecture	Accuracy	F1 Score
CNN (5 s), from scratch	0.81	0.81
CNN (5 s), fine-tuned with 1024 fully-connected units	0.88	0.89
CNN (5 s), fine-tuned with 512 fully-connected units	0.89	0.89
CNN (20 s), from scratch	0.90	0.90
CRNN (5 s), with 5 time steps	0.90	0.91
CRNN (5 s), with 22 time steps	0.90	0.91
CRNN (10 s), for reference	0.98	0.98

Table 8: Various CNN and CRNN model configurations trained on audio samples with durations of 5, 10, and 20 seconds. The ten-second CRNN outperforms the other configurations with an accuracy and F1 score of 0.98 and 0.98, respectively.

Next, we train the same CNN completely from scratch with randomly initialized weights. We had several setbacks when using an Adam optimizer and reverted to using standard SGD to keep the loss value in check. Specifically, we encountered the *exploding gradient problem* [16, p. 288] and had to use *gradient clipping* to solve the issue. With this tweak, we are able to get the model to converge and achieve an accuracy of 0.9090, besting our previous attempts.

Given the larger size of the new dataset, we also tried to increase the number of parameters by doubling the feature maps of the convolutional layers. This, however, did not help. As a next step, we applied the CRNN approach again. Based on this CNN, we added our bidirectional LSTM layers in the same manner as for the previous CRNNs and were able to slightly improve both our accuracy and F1 score to 0.91.

To evaluate how our model architecture fares against established deep learning models, we train a model using Google's Inception-v3 layout [27]. This network design features more layers and is considerable deeper than our proposed CRNN architecture and is the result of Google's latest research into neural networks for image recognition tasks. Instead of a monolithic network design of sequential layers, Inception-v3 uses a combination of parallel inception units, a sequence of convolution layers, as one building block. Evaluating the Inception-v3 model results in a top accuracy of 0.9488, improving our results significantly. Applying the established CRNN treatment to this model increased the performance to an accuracy of 0.9579. In order to boost the performance even further, we also try a CRNN variation, where both the convolutional layer weights and the LSTM weights are trained jointly. In contrast to our initial CRNN approach, this method also updates the existing convolutional filters. We find, however, that this variation does not improve

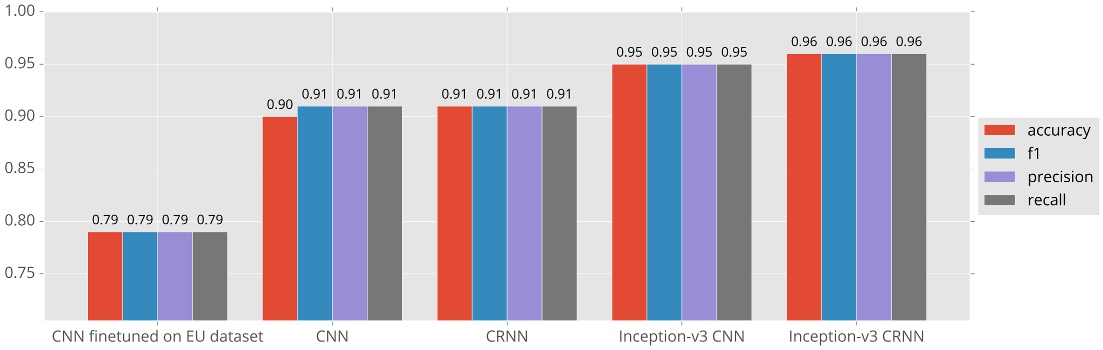


Figure 11: Performance measurement comparison between our CNN and CRNN models and Inception-v3-based models. With a top accuracy of 0.96, the Inception-like CRNN performs best but needs more than five times the number of parameters compared to our proposed model architecture.

performance. Figure 11 shows an overview of the performance metrics of the mentioned experiments. The increased performance, however, does not come without a cost. With a total of 3 153 924 parameters, our CRNN uses roughly six times less parameters than the Inception-v3 CRNN with its 19 million parameters. This increases training time, requires more training data, and consumes more GPU memory. On disk, the serialized model weights come in at 30 MB versus 260 MB, which could be a potential disadvantage for deployment on mobile phones.

7.4.5 Interlanguage Discrimination

Previous work raised concerns about the similarity of our four feature languages—English, German, French, and Spanish—and the model's inability to discriminate between them properly [39]. Both English and German belong to the West Germanic language family, while French and Spanish are part of the Romance languages. We hypothesized that our deep learning approach to language identification is able to differentiate between them reliably.

The results support our hypothesis. Table 9 shows the confusion matrix when evaluating language family pairs on the best-performing CRNN. Spanish and French audio files separate very well with hardly any wrong classifications. Both languages are more likely to be classified as German and English rather than as the respectively other one, demonstrating that their learned representations are quite distinctive within their language family.

German and English language samples have a tendency to be confused. Furthermore, English also has a slight bias towards French, an observation in line with related work [78]. With German, however, the classification error is distributed evenly between French and

	EN	**DE**	**FR**	**ES**
EN	6153	339	181	225
DE	426	6128	173	162
FR	200	145	6447	107
ES	214	170	115	6399

Table 9: Confusion matrix of our best-performing CRNN. Ground truth values are along the rows, predicted values along the columns. English audio files are likely to be misclassified as German and vice versa. Both languages belong the family of Germanic languages.

	EN	**DE**	**FR**	**ES**
EN	6648	140	45	61
DE	152	6639	62	44
FR	68	59	6742	27
ES	75	83	42	6697

Table 10: Confusion matrix of the Inception-v3 CRNN. Ground truth values are along the rows, predicted values along the columns. Despite its deeper architecture, Inception-v3 makes similar types of mistakes as our proposed CRNN.

Spanish. Overall, German samples are misclassified most often across all languages.

The confusion matrix for the Inception-v3 CRNN in Table 10 leads to similar observations as with our CRNN model. Again, German exhibits the most classification errors.

7.4.6 Noise Robustness

Given that we left the raw audio data unchanged, we expected a certain degree of noise within the dataset. Therefore, we hypothesized that the neural network developed some noise robustness by itself. For instance, the convolution operations of the earlier layers summarize pixel values over an image patch and help with masking noise. To prove our theory, we generate two augmented datasets based on the YouTube News dataset.

For the first augmented dataset, we mix the audio signal with randomly generated white noise sounds. The resulting audio samples are still easily identifiable by human listeners.

Dataset	**CRNN**		**Inception-v3 CRNN**	
	Accuracy	**F1 Score**	**Accuracy**	**F1 Score**
No Noise	0.91	0.91	0.96	0.96
White Noise	0.63	0.63	0.91	0.91
Crackling Noise	0.82	0.83	0.93	0.93

Table 11: Accuracy and F1 score for our models evaluated on the speech data augmented with two different types of noise. Our proposed model architecture is susceptible to noise, and its performance deteriorates. The much deeper architecture of Inception-v3 CRNN is more robust against noise and retains a high performance.

The noise has a very strong audible presence, so much that human testers are easily annoyed after a few seconds of listening to it.

For the second augmentation, we add a more periodic crackling noise, emulating analog telephony or a bad voice chat connection. We sample a selection of fitting sound effects and randomly apply these to the source signal using the PyDub library.[15] The resulting noise is not as noticeable and less intrusive as the white noise but gives the augmented audio file a subdued vintage quality.

The white noise significantly deteriorates the language identification performance, both for our CRNN proposal and the Inception-v3 CRNN, as can be seen in Table 11. The noise spectrogram in Figure 12 show that the white noise effect has a very strong influence on the resulting image. Most parts of the image end up covered by the distinct noise texture and only the lower frequency features remain intact. Yet, most speech frequencies are still contained in this range. All pauses and fine-grained details are also lost to the noise sound. The crackling experiment does not incur such a dramatic drop in performance. That might be in part due to the consistent recurring sound of the noise, in contrast to the randomly generated white noise sound. Perhaps, a second factor is the lower, less intrusive volume used for augmenting this dataset.

The deeper, more complex structure of the Inception-v3 CRNN suffers from a significantly smaller performance deterioration than our proposed CRNN model architecture. The more than five-times higher number of parameters seems to capture the frequency features in a more robust manner. In an attempt to remedy the performance loss of our model, we trained and fine-tuned models containing white noise data. We experimented with a 100 % noise dataset and the original YouTube News dataset augmented with 10 % white noise. While both approaches recover some performance in the white noise setting, they impair language identification in the general test set and the crackling noise environment

[15] `https://github.com/jiaaro/pydub`, accessed 01 March 2017

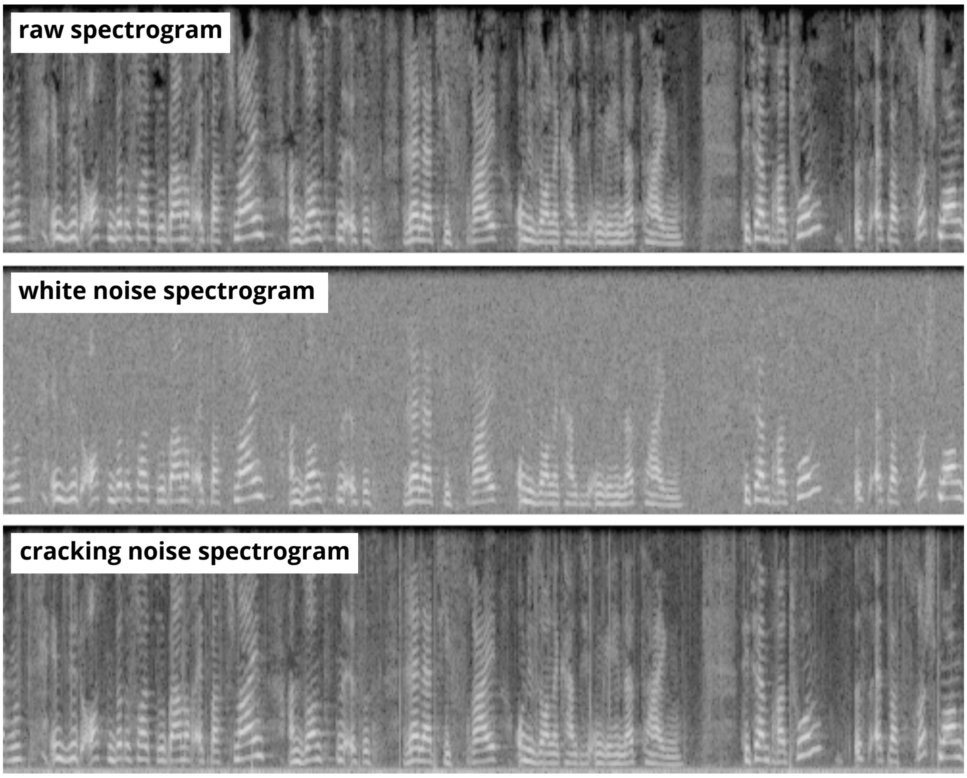


Figure 12: Spectrograms generated from the raw data, augmented with white noise and mixed with a crackling noise emulating analog telephony or a bad voice chat connection. The white noise aggressively subdues most higher frequencies and pauses that decrease the classification performance. The crackling noise is less intrusive and, therefore, does not affect accuracy as much.

as a trade-off.

Let it be noted that our audio mixing was fully automated and that the resulting samples varied in speech and noise volume. We tried to match volume levels of speech and noise to maintain as much clarity of speech as possible, yet some samples have an artificial quality to them.

Overall, we note that even deep convolutional recurrent networks are still subject to the influence of noise on audio. We learned that our spectrogram preprocessing method does not help in these situations, and that different network architectures play an important role in dealing with these situations.

7.4.7 Background Music Robustness

Many real-world audio applications involve some form of music. For example, imagine speaking into your mobile phone from a busy café with background music or having a voice chat session at home while the TV is running. Therefore, we evaluate our model to see if language identification is still possible when mixing speech with music. For this experiment, we use two different test sets. For the first series, we augment our existing YouTube News dataset with randomly sampled background music, similar to the background noise augmentation.

The background audio was obtained from Soundcloud, and features royalty-free, instrument-only music from various genres, including Pop, Dubstep, Rock, and Electro.[16] We normalized the volume of these tracks and overlaid them onto our speech data while trying to stay below the speech audio volume. In the best cases, the two audio streams blend nicely with the background music, producing a soft but noticeable ambient effect while retaining the clarity of the speech. In some other cases, audio level and volume of speech or music are over-accentuated. The final result, however, does neither resemble a song nor a piece of music. We changed neither the tempo nor the pitch of our speakers, and the rhythm of the vocals does not match the rhythm of the background audio.

Given that our training set does not intentionally include samples with music or songs, we expected the performance to be lower than for pure speech samples. Table 12, which shows accuracy and F1 score for the two test sets augmented with background music, confirms this hypothesis. Additionally, it should be noted that the frequency activations of the instruments overlap with the speech frequencies. There is no easy way to separate the instruments from the singer or speaker in a music recording. The approach described here is not intended to work for language identification in songs, and we leave this task open for future work. Figure 13 shows the raw and augmented spectrograms of a Spanish speech snippet. There are several changes of note. First, the large area on the right-hand side,

[16] `https://soundcloud.com/royalty-free-audio-loops`, accessed 01 March 2017

Dataset	**CRNN**		**Inception-v3 CRNN**	
	Accuracy	**F1 Score**	**Accuracy**	**F1 Score**
No Music	0.91	0.91	0.96	0.96
Background Music	0.70	0.70	0.89	0.89

Table 12: Accuracy and F1 score for our models evaluated on the speech data augmented with background music from different genres.

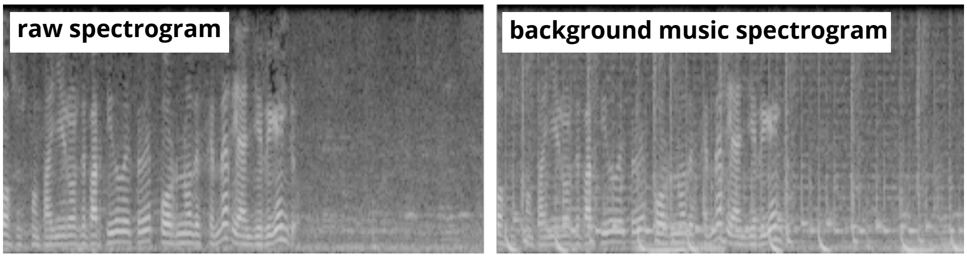


Figure 13: Spectrograms generated from the raw data and augmented with background music noise. The added background music is clearly visible in the resulting spectrogram and changes the classification performance. Note that the previously silent part on the right-hand side now shows frequency activations as well. Additionally, the original ripple-like patterns become a lot more subdued and unclear. Further, new frequency activation patterns along the bottom of the image become visible.

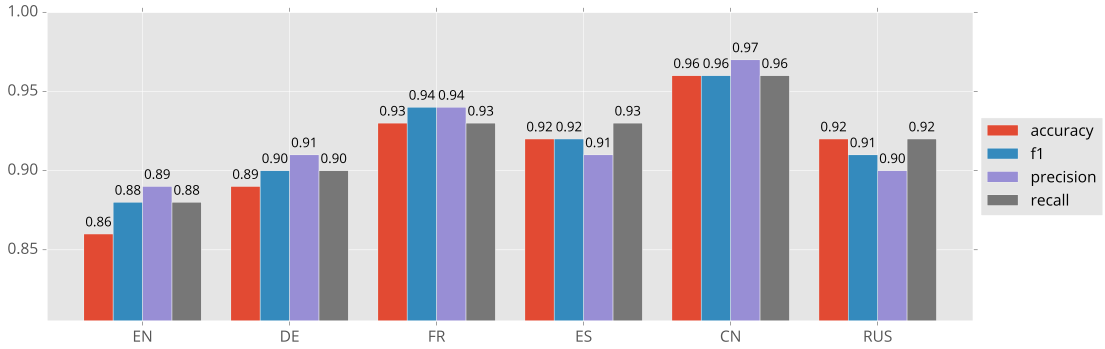


Figure 14: Individual performance measurements for each of our six target languages: English, German, French, Spanish, Mandarin Chinese, and Russian. Chinese exhibits the best misclassification rate, while English performs the worst. Overall, the model performance is consistent with the previous evaluations on only four languages, as highlighted in Section 7.4.4.

which was formerly silent, starts to show repeated patterns. Second, along the lower image boundary, we see new, regular frequency activation spikes. Third, the clear ripple-like patterns in the raw spectrogram turn into muddy, unclear patterns.

7.4.8 Model Extensibility

So far, all our experiments were conducted on datasets consisting of only four languages: English, German, French, and Spanish. We now extend the existing set with two new languages spoken by millions around the globe: Mandarin Chinese and Russian. The goal of this experiment is to learn whether we can expand our model to other languages.

We increase our existing YouTube News dataset with samples taken from Chinese and Russian news channels. Altogether, these now form the Extended YouTube News dataset described earlier in Section 5.3. In order to maintain the class distribution for six languages, we have to decrease the number of training samples of the existing samples slightly. Table 2 contains the details for the Extended YouTube News dataset.

For this experiment, we first fine-tune our previously best CNN by replacing the final fully-connected layers and adjust the number of output nodes to accommodate for six classes. The resulting model serves as the basis for training the CRNN in a similar manner as in earlier experiments. Applied on the test set, we measure an accuracy of 0.92 and an F1 score of 0.92. Both measurements match our previous evaluation with four languages on the YouTube News dataset, proving that the proposed CRNN architecture can be extended to cover more languages. Figure 14 shows individual performance measures for each language. Mandarin Chinese outperforms all other languages with a top accuracy of 0.96, which could be explained by its sound contrast to western languages and its unique intonation. We also note that Russian is most frequently misclassified as Spanish and vice versa. In contrast to our previous observation in Section 7.4.5, German is no longer the worst-performing class but English instead. This is, in part, due to a significant number of misclassifications as Russian samples.

Given that both new languages are rooted within their own respective language families and feature considerably different intonations, we are content to find that the features learned by our model are, indeed, universal in nature. We believe that the approach to language identification proposed in this thesis can be successfully applied to a wide variety of languages.

7.4.9 Visualizations

In all previous sections, our model was described and evaluated by measuring various performance indicators and by applying them to different datasets. In this section, we illustrate earlier observations with plots.

First, we visualize the high-dimensional language vector embedding space using the *t-distributed stochastic neighbor embedding* algorithm (t-SNE) [79]. t-SNE is a nonlinear dimensionality reduction technique employed to map high-dimensional data onto 2D or 3D space for plotting purposes. We apply this machine learning algorithm to the first 2000 predictions of our second-to-last fully-connected layer (right before the classifier) and project our 1024-dimensional YouTube News embeddings into a 2D space. Figure 15 shows the resulting plot, which displays a good separation of our four language classes into independent clusters, confirming that our network learned effective representations of the audio features. Note that French and Spanish are separated very nicely from the other languages, while German and English have some overlap. This is in line with our previous observations of classification errors, as described in Section 5.3.

A primary advantage of deep neural networks is their ability to identify and learn useful features without requiring a data scientist to manually define these. Therefore, no prior domain knowledge is needed, a fact that makes these techniques so powerful and versatile. From an outsider's perspective, deep learning can appear as a black box, and it remains unclear which features were ultimately deemed relevant. In order to gain a better understanding of our model, we visualize its convolutional layers. Figure 16 visualizes nine of the highest-activating filters of the final convolutional layer. To obtain these filters, we performed backpropagation from the output of each filter back to the input image. This yielded the gradients in the filter output with respect to the input image pixels. We used these to perform gradient ascent, searching for the image pixels that maximize the output of the filter, by following the method proposed by Chollet [80].

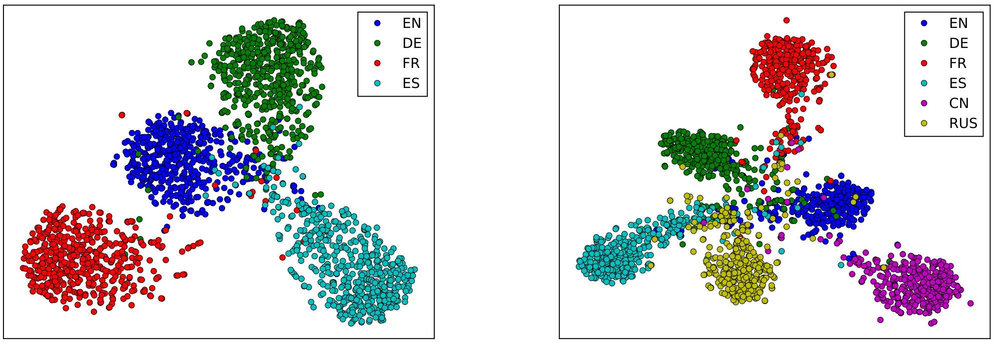


Figure 15: Two-dimensional t-SNE plots of the high-dimensional vector representation of our YouTube News samples for models trained with four and six languages, respectively. All language classes form distinct clusters, confirming that our networks learned effective representations of the audio features.

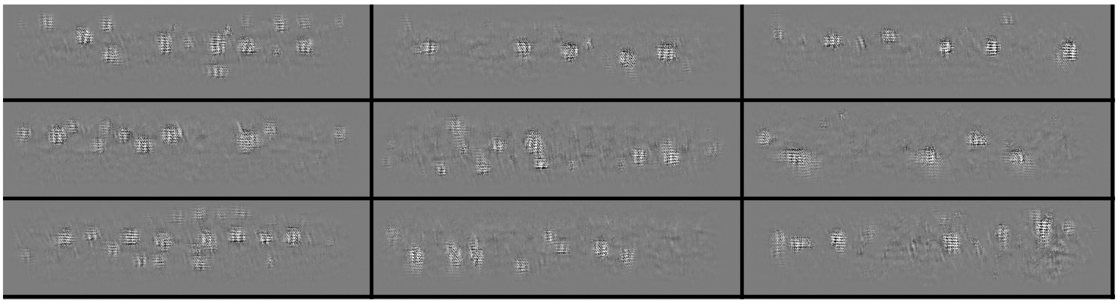


Figure 16: Visualization of nine filters in the final convolutional layer. Note the ripple-like patterns responsible for detecting different frequency spans in the input audio.

Visualizations for the lower-level convolutional layers result in images of very simple geometric shapes like lines and circles, which matched our expectations. With increasing network depth, each convolutional layer's features combined and evolved into more complex shapes resembling our input domain. In Figure 16, we identify the familiar ripple-like patterns that represent frequency activations over time. This proves that the network learned these structures, as we hypothesized earlier. We can also identify that some filters specialize in high frequencies, whereas others focus on lower ones. Furthermore, it can be observed that the filters only react to a short and specific span of time within the spectrogram, all of which are less than one second long.

7.4.10 Discussion

In this thesis, we introduced and compared different CNN and CRNN architectures for language identification and proved that these models can be used to solve our research task with a satisfying quality. Our evaluation confirms that audio tasks can be solved within the image domain using spectrogram image representations. As hypothesized, we were able to prove that our system learned the audio and frequency structures of these spectrogram images. We showed that these results are valid regardless of the input audio source and across various languages. Further, we demonstrated that the learned intermediate language representations were, indeed, universal and not language-specific and could easily be applied to other languages as well.

Our approach of combining convolutional neural networks with bidirectional long short-term memory cells showed a consistent improvement over baseline CNNs. In general, we were able boost accuracy significantly. Furthermore, we established that an Inception-v3-like CRNN outperformed all other approaches, both in terms of accuracy (0.96) as well as noise robustness.

We presented several augmented datasets to evaluate both robustness to noise and background music. We were content to note that the noise resistance of the Inception-v3 CRNN reached acceptable levels without us having to modify or restructure our algorithm. This reaffirms our belief in deep learning techniques as a viable tool for audio tasks.

Our CRNN approach interpreted every vector entry along the x axis as a separate time step for the recurrent layer. Hence, all gathered results are representative of only ten-second snippets of the original audio files. We believe that we could increase the prediction performance in a production-ready system by issueing a majority vote across multiple segments of a longer audio file.

8 Demo Application

To showcase some possible use cases for our language identification system we developed a web service. First, we built a REST interface to our web server that enables a user to upload and identify an audio files. This API could be offered as a standalone service similar to existing speech recognition services such as Google Cloud Speech API[17] or IBM Watson Speech To Text[18]. Secondly, we built a complete website that allows you to upload an audio file for identification. An interactive results page displays the computed prediction probabilities as charts. Further, we embedded a speech to text service that will automatically transcribe your audio file to the identified language. As of May 2017, no commercially available automatic speech recognition service is capable of language identification and hence requires a manual selection of the target language. Our demo application automates this step and preselects the identified language.

The web server was implemented in Python using the Flask[19] micro web development framework. The interactive frontend is designed as a Javascript single page app using the React[20] framework to offer a modular, interactive, and declarative built user interface. The client-side web application architecture is based on the Flux[21] pattern to enforce unidirectional data flow that works well in unison with React's declarative programming style.
The language prediction is handled by the server using Keras and our best performing CRNN model. For each incoming request we apply the same preprocessing steps to the audio file as during our training sessions. Audio files are converted to spectrogram images on the fly before being classified by our deep CRNN system. In contrast to our training data user uploaded audio files can extend past our ten second audio snippet duration. Longer files are split into a sequence of individual ten second snippets for classification. All snippets with a duration of less than ten seconds are discarded. The resulting prediction probabilities are averaged into a single fusion score.

image

[17] https://cloud.google.com/speech, accessed 16.05.2017
[18] https://www.ibm.com/watson/developercloud/speech-to-text.html, accessed 16.05.2017
[19] http://flask.pocoo.org, accessed 16.05.2017
[20] https://facebook.github.io/react/, accessed 16.05.2017
[21] https://facebook.github.io/flux, accessed 16.05.2017

9 Conclusion and Future Work

This last chapter presents high-level insights and finally comments on future work and prospects concerning the discussed topics. It further outlines the contribution of this thesis and concludes the work done in the research field.

9.1 Future Work

The overall trend within the deep learning community over the past few years has been to increase the network depth by adding more and more layers. Supporting this development, however, is becoming increasingly more difficult. Very deep networks are harder to train and require ever more computing time. One way to gain higher accuracy scores is to change the design and architecture of the network. He et al. recently proposed the technique of deep residual learning[13] to tackle these challenges. In their network architecture they introduce small building blocks of two convolutional layers in which the second layer receives the original input of the first layer in addition to the output of its predecessor. [13] Using this approach, they were able to train *ResNet* with 152 layers and bested the previous state of the art in the ILSVRC 2015 challenge. Similarly, Szegedy et al. introduced the latest iteration of their Inception networks using these residual connections[28] as well.
For future work, we think that using deep residual neural network for the convolutional part of our architecture holds much promise. In this thesis we were able to measure our best results using the previously state-of-the-art *Inception-v3* network for computer vision tasks. In the future we believe that using *Inception-v3* or the ResNet architecture should improve our results even further and perhaps add more robustness to noise as well.

As part of our training process, we pretrained our convolutional neural networks before assembling them to the complete CRNN. The complete CRNN then reused the learned weights of the convolutional layers and finetuned them by further, joint training with the LSTM part. This *transfer learning* approach is fairly simple, yet effective. Unfortunately, parallel to finetuning the network on the new data it starts to forget previously learned features. This is referred to as catastrophic forgetting. Rusu et al. introduced *Progressive Neural Networks*[81] as a novel approach to *transfer training.* This technique leverages prior knowledge via so called lateral connections to previously learned features. For future work, we think it is worthwhile to evaluate whether this approach could be helpful in improving our transfer learning steps.

We formulated our research problem as a classification problem for language identification. The limitation of this, however, is that our models are only able to accurately predict languages that have been part of the training corpus. In other words, we are unable to identify any language unknown to the system. An interesting, yet slightly different research field, is metric learning. This approach is able to measure a difference or a score between its classes. So for example with metric learning, a sample would receive a score whether it was closer related to German or English. The advantage of this approach is that the system is no longer limited to only the training languages. For unknown languages one could still assess, whether a language is more similar to one then another. Conversely, this also allows to judge how different an input is to any language know to the system, something that is not possible with our approach.

In this thesis we evaluated our networks on six different input languages. In the future our system could be extended to cover even more languages. In our work we did not evaluate the effectiveness of the presented approach on similar tasks such as dialect identification. Future work could evaluate the accuracy of our approach on the fine grained differences between dialects and accents of a language.

Throughout this work we evaluated our models' robustness to white noise and background music noise. We also tried evaluating our models on songs but found the performance to be inadequate for our needs. In our case the biggest problem with songs was the fact that the speech frequencies are completely masked by the frequencies of the different instruments. Future work could focus on language identification for songs and music.

Our approach relies on extracting and classifying a sequence of intermediate languages representation created from spectrogram image inputs. Within the automatic speech recognition (ASR) community there is related work[37]that extracts and classifies a sequence of phonemes. Instead of matching these phoneme sequences to words one could use these to classify a target language. This is an alternative approach to the one presented in this thesis and could be used to compare the effectiveness and robustness of the two methods. Another interesting idea was introduce by Google's *Wavenet*[82]. Utilizing dilated convolutions for speech recognition they were able to capture a longer receptive field while being much cheaper to compute than LSTMs. Further, *Wavenet* operates on raw audio signals forgoing the need for spectrogram features. Even though they evaluated *Wavenet* for automatic speech recognition, it could likely be adopted to language identification as well.

9.2 Conclusion

In this thesis we presented several neural network architectures aimed at the task of solving the language identification problem. We employed CNNs and CRNNs to infer the language of a given audio sample directly from its spectrogram representation.

We proposed three hypotheses and conducted experiments to verify their validity. We confirmed that, first convolutional neural networks are an effective, high-accuracy solution for language identification tasks. Second, spectrogram images are a suitable input representation for learning audio features. Third, convolutional recurrent neural networks consistently improve the classification accuracy throughout all experiments compared to a plain, CNN-based approach.

We present and discuss the availability and suitability of various datasets for LID. For our research we compiled and processed more than a thousand hours of speech data from public sources. On the one hand, this thesis gathered speeches and press conferences from the European Parliament. On the other hand, we collected audio from news broadcasts hosted on YouTube.

To judge the effectiveness and validity of our research in several scenarios, we augmented our test dataset with additional white noise, crackling noise, and background music and tested the robustness of our neural networks. We report a decrease in recognition accuracy under these noisy situations.

In this work, we explored many different neural network architectures and comment on our setup of hyperparameters. Our best performing CRNN model is based the Inception-v3 architecture and was able to achieve an accuracy and F1 score of 0.96 and 0.96, respectively.

Initially, we trained and evaluated our models on four languages—English, German, French, and Spanish. We were content to find that the approach presented in this thesis could be transferred to other languages as well. During our experiments we added Mandarin Chinese and Russian. The frequency features learned by our system are, indeed, language-independent and we were able to expand our system to these two new languages without any large modifications.

This thesis presents a recent approach to solve the language identification task with deep neural networks. Further research needs to be deducted to improve the robustness against noise and discover new kinds of layers and architectures to support even higher accuracies in this field.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[2] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[3] H. Yang, C. Wang, C. Bartz, and C. Meinel, “Scenetextreg: A real-time video ocr system,” in *ACM Multimedia*, 2016.

[4] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, “Synthetic data and artificial neural networks for natural scene text recognition,” *arXiv preprint arXiv:1406.2227*, 2014.

[5] J. Dai, K. He, and J. Sun, “Instance-aware semantic segmentation via multi-task network cascades,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3150–3158, 2016.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[7] H. Nam and B. Han, “Learning multi-domain convolutional neural networks for visual tracking,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4293–4302, 2016.

[8] G. Tolias, R. Sicre, and H. Jégou, “Particular object retrieval with integral max-pooling of cnn activations,” *arXiv preprint arXiv:1511.05879*, 2015.

[9] Y. K. Muthusamy, E. Barnard, and R. A. Cole, “Reviewing automatic language identification,” *IEEE Signal Processing Magazine*, vol. 11, no. 4, pp. 33–41, 1994.

[10] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, “Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1,” *NASA STI/Recon technical report n*, vol. 93, 1993.

[11] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[12] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[14] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[15] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[18] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1891–1898, 2014.

[19] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification.," in *EMNLP*, pp. 1422–1432, 2015.

[20] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[21] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, 2012.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[24] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[26] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, “Return of the devil in the details: Delving deep into convolutional nets,” in *British Machine Vision Conference*, 2014.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[28] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *arXiv preprint arXiv:1602.07261*, 2016.

[29] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, 2012.

[30] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[31] B. Shi, X. Bai, and C. Yao, “An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[32] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,” in *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376, ACM, 2006.

[33] J. Gonzalez-Dominguez, I. Lopez-Moreno, H. Sak, J. Gonzalez-Rodriguez, and P. J. Moreno, “Automatic language identification using long short-term memory recurrent neural networks,” in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[34] K. J. Han, S. Ganapathy, M. Li, M. K. Omar, and S. Narayanan, “Trap language identification system for rats phase ii evaluation.,” in *INTERSPEECH*, pp. 1502–1506, 2013.

[35] P. Matejka, L. Zhang, T. Ng, H. S. Mallidi, O. Glembek, J. Ma, and B. Zhang, "Neural network bottleneck features for language identification," *Proc. IEEE Odyssey*, pp. 299–304, 2014.

[36] F. Richardson, D. Reynolds, and N. Dehak, "A unified deep neural network for speaker and language recognition," *arXiv preprint arXiv:1504.00923*, 2015.

[37] W. Song and J. Cai, "End-to-end deep neural network for automatic speech recognition," tech. rep., Technical Report CS224D, University of Stanford, 2015.

[38] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," *arXiv preprint arXiv:1512.02595*, 2015.

[39] G. Montavon, "Deep learning for spoken language identification," in *NIPS Workshop on deep learning for speech recognition and related applications*, pp. 1–4, 2009.

[40] S. Dieleman and B. Schrauwen, "Multiscale approaches to music audio feature learning," in *14th International Society for Music Information Retrieval Conference (ISMIR-2013)*, pp. 116–121, Pontifícia Universidade Católica do Paraná, 2013.

[41] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in neural information processing systems*, pp. 1096–1104, 2009.

[42] J. Wülfing and M. A. Riedmiller, "Unsupervised learning of local features for music classification.," in *ISMIR*, pp. 139–144, 2012.

[43] M. Henaff, K. Jarrett, K. Kavukcuoglu, and Y. LeCun, "Unsupervised learning of sparse features for scalable audio classification.," in *ISMIR*, vol. 11, p. 2011, 2011.

[44] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011.

[45] D. Garcia-Romero and C. Y. Espy-Wilson, "Analysis of i-vector length normalization in speaker recognition systems.," in *Interspeech*, vol. 2011, pp. 249–252, 2011.

[46] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 6964–6968, IEEE, 2014.

[47] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *arXiv preprint arXiv:1609.03193*, 2016.

[48] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, *et al.*, "Recent advances in deep learning for speech research at microsoft," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8604–8608, IEEE, 2013.

[49] A. Sizov, K. A. Lee, and T. Kinnunen, "Discriminating languages in a probabilistic latent subspace," in *Odyssey: the Speaker and Language Recognition Workshop*, 2016.

[50] D. Martınez, O. Plchot, L. Burget, O. Glembek, and P. Matejka, "Language recognition in ivectors space," *Proceedings of Interspeech, Firenze, Italy*, pp. 861–864, 2011.

[51] O. Plchot, P. Matejka, R. Fér, O. Glembek, O. Novotnỳ, J. Pešán, K. Veselỳ, L. Ondel, M. Karafiát, F. Grézl, *et al.*, "Bat system description for nist lre 2015," in *Proc. Odyssey*, 2016.

[52] J. Gonzalez-Dominguez, I. Lopez-Moreno, P. J. Moreno, and J. Gonzalez-Rodriguez, "Frame-by-frame language identification in short utterances using deep neural networks," *Neural Networks*, vol. 64, pp. 49–58, 2015.

[53] G. Gelly, J.-L. Gauvain, L. Lamel, A. Laurent, V. B. Le, and A. Messaoudi, "Language recognition for dialects and closely related languages," *Odyssey, Bilbao, Spain*, 2016.

[54] "The 2015 nist language recognition evaluation plan (lre15)." `https://www.nist.gov/sites/default/files/documents/2016/10/06/lre15_evalplan_v23.pdf`, 2015.

[55] K. Lee, H. Li, L. Deng, V. Hautamäki, W. Rao, X. Xiao, A. Larcher, H. Sun, T. Nguyen, G. Wang, *et al.*, "The 2015 nist language recognition evaluation: the shared view of i2r, fantastic4 and singams," in *Interspeech 2016*, vol. 2016, pp. 3211–3215, 2016.

[56] P. A. Torres-Carrasquillo, E. Singer, W. M. Campbell, T. P. Gleason, A. McCree, D. A. Reynolds, F. Richardson, W. Shen, and D. E. Sturim, "The mitll nist lre 2007 language recognition system.," in *Interspeech*, pp. 719–722, Citeseer, 2008.

[57] R. W. Ng, M. Nicolao, O. Saz, M. Hasan, B. Chettri, M. Doulaty, T. Lee, and T. Hain, "The sheffield language recognition system in nist lre 2015," in *Proceedings of The Speaker and Language Recognition Workshop Odyssey 2016*, pp. 181–187, ISCA, 2016.

[58] R. Zazo, A. Lozano-Diez, and J. Gonzalez-Rodriguez, "Evaluation of an lstm-rnn system in different nist language recognition frameworks," *Odyssey 2016*, pp. 231–236, 2016.

[59] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition.," in *INTERSPEECH*, pp. 3586–3589, 2015.

[60] X. Cui, V. Goel, and B. Kingsbury, "Data augmentation for deep neural network acoustic modeling," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 23, no. 9, pp. 1469–1477, 2015.

[61] N. Jaitly and G. E. Hinton, "Vocal tract length perturbation (vtlp) improves speech recognition," in *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language*, 2013.

[62] E. Eide and H. Gish, "A parametric approach to vocal tract length normalization," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 346–348, IEEE, 1996.

[63] H. Kantilaftis, "How to read the news like a professional news anchor." `https://www.nyfa.edu/student-resources/how-to-read-the-news-like-a-professional-news-anchor/`, 2016.

[64] E. Charniak, D. Blaheta, N. Ge, K. Hall, J. Hale, and M. Johnson, "Bllip 1987-89 wsj corpus release 1," *Linguistic Data Consortium, Philadelphia*, vol. 36, 2000.

[65] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 5206–5210, IEEE, 2015.

[66] J. Gonzalez-Dominguez, D. Eustis, I. Lopez-Moreno, A. Senior, F. Beaufays, and P. J. Moreno, "A real-time end-to-end multilingual speech recognition architecture," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 749–759, 2015.

[67] F. Chollet, "Keras." `https://github.com/fchollet/keras`, 2015.

[68] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[69] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[70] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python." `http://www.scipy.org/`, 2001.

[71] R. B. Blackman and J. W. Tukey, "The measurement of power spectra from the point of view of communications engineering - part 1," *Bell Labs Technical Journal*, vol. 37, no. 1, pp. 185–282, 1958.

[72] H. Traunmüller and A. Eriksson, "The frequency range of the voice fundamental in the speech of male and female adults," 1993.

[73] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of cnn advances on the imagenet," *arXiv preprint arXiv:1606.02228*, 2016.

[74] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[75] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[76] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, vol. 9, pp. 249–256, 2010.

[77] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[78] T. Werkmeister and T. Herold, "Practical applications of multimedia retrieval: Language identification in audio files." `https://github.com/twerkmeister/iLID/blob/master/Deep%20Audio%20Paper%20Thomas%20Werkmeister%2C%20Tom%20Herold.pdf`, 2016.

[79] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[80] "How convolutional neural networks see the wolrd." `https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html`, 2016.

[81] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[82] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR abs/1609.03499*, 2016.